Provably Secure Authenticated Encryption

THÈSE N^{O} 8681 (2018)

PRÉSENTÉE LE 31 AOÛT 2018

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE DE SÉCURITÉ ET DE CRYPTOGRAPHIE
PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Damian VIZÁR

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2018

Túto prácu venujem svojim rodičom

Abstract

Authenticated Encryption (AE) is a symmetric key cryptographic primitive that ensures confidentiality and authenticity of processed messages at the same time. The research of AE as a primitive in its own right started in 2000.

The security goals of AE (such as NAE, MRAE, OAE, RAE or the RUP) were captured in formal definitions in the tradition *provable security*, where the security of a scheme is formally proven assuming the security of an underlying building block. The prevailing syntax moved to *nonce-based* AE with *associated data* (which is an additional input that gets authenticated, but not encrypted). Other types of AE schemes appeared as well, e.g. ones that supported stateful *sessions*.

Numerous AE schemes were designed; in the early years, these were almost exclusively blockcipher modes of operation, most notably OCB in 2001, CCM in 2003 and GCM in 2004. At the same time, issues were discovered both with the security and applicability of the most popular AE schemes, and other applications of symmetric key cryptography.

As a response, the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) was started in 2013. Its goals were to identify a portfolio of new, secure and reliable AE schemes that would satisfy the needs of practical applications, and also to boost the research in the area of AE. Prompted by CAESAR, 57 new schemes were designed, new types of constructions that gained popularity appeared (such as the Sponge-based AE schemes), and new notions of security were proposed (such as RAE). The final portfolio of the CAESAR competition should be announced in 2018.

In this thesis, we push the state of the art in the field of AE in several directions. All of them are related to provable security in one way or another.

We propose OMD, the first provably secure dedicated AE scheme that is based on a compression function. We further modify OMD to achieve nonce misuse-resistant security (MRAE). We also propose another provably secure variant of OMD called pure OMD, which enjoys a great improvement of performance over OMD.

Inspired by the modifications that gave rise to pure OMD, we turn to the popular Sponge-based AE schemes and prove that similar measures can also be applied to the keyed Sponge and keyed Duplex (a variant of the Sponge), allowing a substantial increase of throughput without an impact on security.

We then address definitional aspects of AE. We critically evaluate the security notion of

OAE, whose authors claimed that it provides the best possible security for online schemes under nonce reuse. We challenge these claims, and discuss what are the meaningful requirements for online AE schemes. Based on our findings, we formulate a new definition of online AE security under nonce-reuse, and demonstrate its feasibility.

We next turn our attention to the security of nonce-based AE schemes under *stretch misuse*; i.e., when a scheme is used with varying ciphertext expansion under the same key, even though it should not be. We argue that varying the stretch is plausible, and formulate several notions that capture security in presence of variable stretch. We establish their relations to previous notions, and demonstrate the feasibility of security in this setting.

We finally depart from provable security, with the intention to complement it. We compose a survey of universal forgeries, decryption attacks and key recovery attacks on 3rd round CAESAR candidates.

Keywords: Authenticated Encryption, Provable Security, Misuse Resistance, CAESAR competition, Compression Function, Sponge Construction, Full-state Absorption, Online Authenticated Encryption, Variable Tag Length, Cryptanalysis

Abstrakt

Authentifizierte Verschlüsselung (AE, aus dem englischen “Authenticated Encryption”) ist ein symmetrisches Kryptosystem das gleichzeitig Vertraulichkeit, Authentizität und Integrität sicherstellt. Die Forschung an AE als einer eigenständigen Verschlüsselungsart begann im Jahr 2000.

Die Sicherheitsziele von AE wurden in formalen Definitionen in der Tradition der *beweisbaren Sicherheit* festgehalten (wie zum Beispiel MRAE, OAE, RAE oder die RUP-Begriffe). Die vorherrschende Syntax wurde zu Nonce-basierender AE mit *zugehörigen Daten* verschoben. Die zugehörige Daten sind eine zusätzliche Eingabe, die authentifiziert, aber nicht verschlüsselt wird. Andere Arten von AE-Schemata erschienen, zum Beispiel diejenigen, die *Sitzungen* unterstützten.

Zahlreiche AE-Schemata wurden entworfen; in den Anfangsjahren waren dies fast ausschließlich Betriebsmodi von Blockchiffren. Beispiele solchen Betriebsmodi sind OCB aus dem Jahr 2001, CCM vorgestellt im Jahr 2003 und GCM von 2004. Gleichzeitig wurden Probleme mit der Sicherheit und Anwendbarkeit der beliebtesten AE-Schemata und anderen Anwendungen der Kryptographie mit symmetrischen Schlüsseln entdeckt.

Im Jahr 2013 wurde der CAESAR Wettbewerb (aus dem englischen “Competition for Authenticated Encryption: Security, Applicability, and Robustness”) gestartet. Sein Ziel war es, ein Portfolio neuer, sicherer, praktischer und zuverlässiger AE-Schemata zu identifizieren. Auf Anregung von CAESAR wurden 57 neue Schemata entworfen, neue Konstruktionen, die an Popularität gewannen (wie die sogenannten Sponge-basierten AE-Systeme), erschienen, und neue Sicherheitsbegriffe (wie RAE) wurden vorgeschlagen. Das endgültige Portfolio des CAESAR-Wettbewerbs sollt im Jahr 2018 bekannt gegeben werden.

In dieser Arbeit treiben wir den Stand der Technik in dem Gebiet der AE in mehreren Richtungen voran. Alle präsentierten Ergebnisse beziehen sich auf die eine oder andere Weise auf die nachweisbare Sicherheit.

Wir schlagen OMD vor, das erste beweisbar sichere dedizierte AE-Schema, das auf einer Komprimierungsfunktion basiert. Wir modifizieren OMD weiter, um Nonce Missbrauch-Widerstandsfähigkeit zu erreichen. Wir schlagen auch eine andere beweisbar sichere OMD-Variante vor, die eine deutliche Verbesserung der Leistung gegenüber OMD bietet.

Inspiriert von den Modifikationen, die zu “pure OMD” geführt haben, beweisen wir, dass ähnliche Maßnahmen auch auf den beliebten “keyed Sponge” und “keyed Duplex” (eine Variante der Sponge-konstruktion) angewendet werden können, die ihre Leistung verbessern, ohne Auswirkungen auf die Sicherheit zu haben.

Wir bewerten dann den Sicherheitsbegriff OAE kritisch. Seine Authoren behaupteten, dass OAE die bestmögliche Sicherheit für Online-Schemata unter Nonce-Wiederverwendung bietet. Wir stellen diese Behauptung in Frage und formulieren eine neue Definition der Online-AE-Sicherheit unter Nonce-Missbrauch-Widerstandsfähigkeit.

Wir lenken dann unsere Aufmerksamkeit auf die Sicherheit von Nonce-basierten AE-Schemata unter “*Stretch Missuse*.” Wir argumentieren, dass “Stretch Missuse” plausibel ist und formulieren mehrere Begriffe, die Sicherheit bei “Stretch Misuse” erfassen.

Wir verlassen schließlich die beweisbare Sicherheit, um sie zu ergänzen. Wir stellen eine Übersicht der universellen Fälschungen, Entschlüsselungsangriffen und Brute-Force-Angriffen auf Kandidaten in der dritten Runde CAESARs zusammen.

Schlüsselwörter: Authentifizierte Verschlüsselung, Beweisbare Sicherheit, CAESAR Wettbewerb, Vollzustandsabsorption, Missbrauch-Widerstandsfähigkeit, Sponge, Komprimierungsfunktion, Online Authentifizierte Verschlüsselung, Variable Taglänge, Kryptoanalyse

Acknowledgements

Right after my PhD defense,[1] many of you asked me how does it feel to have successfully defended, and being just a half-step away from finally becoming a doctor. The truthful answer that I kept returning to that question was "the same." After all, there was nothing in my life that radically changed during those 135 minutes. Yet, as I am pondering my next words, my answer changes to "amazed," because I realize how much has changed during those five years—it was quite a journey! And I would have never been able to make that journey just by myself.

First, I would like to thank my supervisor Prof. Serge Vaudenay. Serge is, in my opinion, one of the best PhD advisers one can wish for, and there are a few things which I really appreciate about him. He was sufficiently strict, especially at the beginning of my PhD, which made me realize (and reminded me a few times), that the main ingredient of successful doctoral studies is hard work. He was always available, not just for me, but for any member of the lab; no mater how busy he was, he always found some time to discuss a difficult proof, fuzzy formalism or a job application. And, what is quite amazing, he was always able to provide insight and concrete technical advice for any possible topic. At the same time, I had a lot of freedom in my research and collaborations which is not always granted. On top of this, Serge was also genuinely interested in our lives beyond our work, which is also not always granted. Thank you Serge, for all this, and all your help and guidance!

I would like to thank the members of my committee. I thank Prof. Arjen Lenstra for chairing my private defense as the committee president, and I thank Prof. Carmela Troncoso for serving as the internal reviewer of my thesis. I would like to thank Prof. Joan Daemen and Prof. Phillip Rogaway for joining my committee as external reviewers, and making the trip to Lausanne to attend the defense in person—I appreciate that a lot. I would like to thank all of my committee members for their insightful, and accurate comments, which helped me critically place my work into a broader perspective.

I would like to thank our secretary Martine Corval, without whom my life in Lausanne would surely be much more complicated. She was taking care of all the imaginable administrative work to allow us to focus on our research an teaching. She helped me

[1]Right after the *private* defense, to be precise.

whenever I had to deal with an incomprehensible administrative letter in French, a difficult phone call or when I had to navigate through a complicated situation involving the official authorities, the management of EPFL or the rental agencies. No matter how much work was on her plate, she always found a few minutes for our friendly morning chat in French. Thank you Martine!

Next, I would like to thank my two office mates Handan Kılınc and Petr Sušil. Handan shared an office with me during the last three years of my PhD. Her cheerful greetings, our ad-hoc conversations over the computer screens, and the exchange of snacks made the everyday life at work nicer. She was also very tolerant to my tendency to create chaos in my half of the office. She was a fellow TA with me in the Cryptography and Security course for three years, and a fellow conspirator of the birthday cards and cakes, and the person who watered my plants when I was travelling. We've been through a lot together! Thank you Handan.

Petr was my office mate for one year before Handan. During this short time, my scripting skills in Bash, and my Linux administration skills increased exponentially, thanks to him. As a senior PhD, he also bestowed a lot of wisdom upon me, concerning the life at EPFL and the life in Lausanne, such as the famous Sušil's principle of elevation-conservation. Petr also taught me (by example) that sometimes not worrying about problems too much is the easiest way to their solution. It was very nice to have someone who comes from the same culture, speaks ≈the same language[2] and understands the cryptographer's inside jokes at the same time.

Big thanks goes to Alexandre Duc and Sonia Bogos,[3] who made me quickly feel like a part of the lab when I started, and became my friends. They initiated me into the secretive group known as NSA at EPFL (although I turned out to be a rather bad apprentice). They brightened every afternoon with a piece of great Swiss chocolate, tirelessly answered all of my numerous questions I had at the beginning of my PhD, never hesitated to help me out with technical questions, and also taught me (by example) how to be a good teacher. They also taught me how to discover and get free food and services, which is an indispensable skill for a PhD student. We enjoyed numerous hikes, games-nights and a lot of "advanced fun" together. Thank you guys!

I would like to thank Fatih Balli, a fellow cyclist, a connoisseur of nerdy jokes, and the most likely target of my occasional trolling (to his credit, he always takes it in good humor) for interesting conversations, especially those seeking enlightenment in the form of a perfect formalism. I would also like to thank Gwangbae Choi, a fellow TA who possesses wisdom beyond his years for sharing his insights about life. I would like to thank Betül Durak for morning conversations, for sharing silly jokes, and for taking the initiative in organizing lab events.

[2]Here, Petr would argue that Slovak is just a dialect of Czech.

[3]Special acknowledgement goes to Sonia here, who was my official EPFL Buddy at the beginning of my studies.

I would also like to thank all the other colleagues from LASEC for being steadfast companions on this journey: Iraklis Leontiadis, Subhadeep Banik, Divesh Aggarwal, Adeline Langlois, Sebastian Faust, Aslı Bay, Florian Tramèr and Jialin Huang. I would like to thank all the summer interns that passed through LASEC for making making the summer months at EPFL fun and diverse.

I would also like to thank Robin and Ralph Ankele, Johan Droz and Martin Georgiev for their work on the implementations of OMD that also contributed to the contents of this thesis, and all other students who worked on the semester projects that I proposed. Thank you for your work guys!

I would like to especially thank Reza Reyhanitabar, who is a co-author in most of my papers, who helped me set off on the research path that I have taken, and who was the most involved in mentoring my early work. Thank you Reza for sharing your knowledge and experience, and teaching me valuable lessons about how to deal with paper submissions and project reports.

Next, I would like to thank my co-authors who have not yet been listed as such: Phillip Rogaway, Tung Hoang, Bart Mennink, Tetsu Iwata, Guillaume Endignoux, Davide Naccache, Simon Cogliani, Diana Maimut and Rodrigo Portella do Canto. Thank you for the collaborations and for your insights. I would like to thank especially Phil, Tung and Bart: I learned a lot from each of you, and each of you helped this thesis to take its present shape.

My thanks go to the fellow PhDs who started in the same “batch” as myself, and who became my friends here in Lausanne: Anh, Victor, Giel, Jonas, Adi, Catherine, Julia, Sam, George (Prekas), Timur, Amit, and George Psaropoulos. Special shoutout goes to Jonas, Adi and Giel, who comprise the reliable beer-brigade. A year ago, Giel traded the imaginary lab coat for a white collar. The spontaneous outings in his company have been sorely missed since then.

I would like to thank my Slovak friends who I got to know here in Lausanne. Thanks to Matej, for being an amazingly selfless and cheerful friend, an expert bike mechanic, a bike-pimp and a ski-touring liaison in one person. Thanks to Fero, Rado and Paľo, who together with Matej form a strong group with which we enjoyed a lot of fun and special jokes. Thanks to Katka and Ondro, who were keeping up the semi-regulars meetings of the Slovak group while they were in Lausanne, and are still remotely organizing occasional reunions remotely from Zurich. Thanks to Jakub and Terka, our Slovak-Swiss-Spanish friends for chill movie evenings and interesting conversations about entrepreneurship.

I would also like to thank my dear friends back home in Slovakia, with whom I can always meet, talk and make stupid jokes as if we just saw each other yesterday, no matter for how long we have actually not seen each other. Their friendship is one of the things that make me feel at home when I go back home. Thanks to Tošo, Duďo, Roman, Fipo, Škrabal, Zdany, Muco, Lienka, Gretel, Erika, Mat'a, Lubi and Ninja.

I would like to send a big thanks to my parents, for all their support and their unconditional parental love. They always did everything they thought could help me, and let me do what I deemed fit (even though they sometimes quite expressly disagreed). It was their care and support that sent on the journey towards my PhD, and they did everything to be here for me during the PhD. They shared all the worries and joys I had during my years at EPFL and visited every year, so sometimes it felt as though we were not so far from each other.

I would like to thank my brother Martin and his girlfriend Miška who always go out of their way to spend time with us, whenever they have an opportunity.

I would also like to thank the rest of my (big extended) family, whom it is always a pleasure to meet, for being constantly supportive.

Finally, I would like thank my dearest Katarína, for encouraging me when I feel down, for being patient when I am being petulant, for calming me down when I am panicking and simply for always being that what I am missing. She has been with me throughout my journey through the PhD studies, and now we are setting off on a new, perhaps the most exciting journey of them all.

Table of Contents

Chapter 1

Introduction

The subject of this thesis lies in the field of cryptography. The word cryptography originates from the Greek words *kryptos* (meaning “hidden”) and *graphein* (meaning “to write”) [Mer09]. This etymology fittingly describes the goal of ancient cryptography: to ensure *confidentiality* of information when facing an *adversary* who may eavesdrop on the channel through which the information is being sent. This had been its only goal from the time of ancient “secret codes” and ciphers that appeared in the Hebrew, Egyptian, Mesopotamian or Greek cultures, all the way to the early 20th century, when the arts[1] of cryptography and cryptanalysis suddenly gained a lot more importance, due to the invention of radio communication [Kah96].

In 1945, Claude Shannon wrote the paper titled “A Mathematical Theory of Cryptography” [Sha45] (the paper was publicly disclosed in 1949 [Sha49]), which is arguably one of the cornerstones of cryptography as a scientific discipline. In it, Shannon presented the first mathematical model of cryptosystems coupled with a formal definition of their security. Shannon’s work focused on *secrecy*, i.e., confidentiality [Sti95, Bau01]. Some 35 years later, another fundamental goal of modern cryptography received formal treatment when G. J. Simmons presented his theory for ensuring *authenticity* (i.e., correct determination of the origin) of information [Sim84]. Closely related to authenticity, and sometimes conflated with it, is the goal of ensuring *integrity* of information, i.e., preventing information from being maliciously modified by an adversary.

Originally, all of cryptographic work was done in the *symmetric key* setting: for example, to enforce confidentiality of their communication, two parties, call them Alice and Bob, first exchange a secret value, called *secret key.* Alice then applies a symmetric encryption scheme that maps each message (also called *plaintext)* to a ciphertext using the secret key, such that only a person in possession of the same secret key will be able to extract the original messages from the ciphertexts. An example of an early symmetric encryption primitive is the Data Encryption Standard [Des77].

This changed when Diffie and Hellman introduced the concept of *public key* crypto-

[1]Cryptography could not yet be considered a science.

graphy in their seminal paper from 1976 [DH76], which eliminated the need for cumbersome exchange of secret keys. Soon after, Rivest, Shamir and Adleman designed the first public key cryptosystem [RSA78].

In 1982, Goldwasser and Micali proposed the notion of *semantic security* for public key cryptosystems [GM82], which captured a strong but natural form of confidentiality protection in a precise, formal definition. They further proposed the first probabilistic cryptosystem and *proved* that it is semantically secure under the assumption that the problem of deciding quadratic residuosity (in a specific type of rings of modular integers) is hard. Thus they laid foundations of the rigorous research methods called *provable security*.[2] In 1997, Bellare, Desai, Jokipii and Rogaway extended the provable security treatment to symmetric encryption [BDJR97].

In the last four decades, cryptography has evolved significantly. On one hand, the way cryptography is used changed: while at the beginning of the 20th century, the almost exclusive users of cryptography were military and intelligence agencies, today everyone is (unknowingly) using cryptography, which is ubiquitous in modern technology [KL14]. On the other hand, the field of cryptography itself grew and developed. The list of goals cryptography now seeks to achieve has grown considerably, well beyond the fundamental trilogy of confidentiality, integrity and authenticity. Today it is extended by privacy,[3] non-repudiation, unpredictability, or fair termination, just to name a few.

Even though there are so many different security goals to pursue, and despite the fact that the public key cryptography is more convenient to set up, the symmetric key cryptographic primitives that ensure confidentiality, authenticity and integrity are still among very active research topics of modern cryptography, mainly because of their relevance to practice.

Before authenticated encryption. Achieving confidentiality and authenticity/integrity[4] by symmetric key primitives belongs to the most fundamental, and oldest topics in cryptography. They were traditionally studied and achieved separately.

Confidentiality has mainly been achieved with help of *streamciphers* and *blockciphers*. Streamciphers are inspired by the one-time pad.[5] A streamcipher, such as RC4 or ChaCha [Ber08], takes a fixed-size secret key and a fixed-size, non-repeating initialisation vector (a *nonce*) as inputs, and generates a pseudorandom string of (practically) arbitrary length that is used to "mask" a message, typically by the means of bitwise XOR. A blockcipher maps a fixed-size plaintext and a fixed-size secret key to a fixed-size output block. The most notable example for a blockcipher is AES [DR02, Pub01]. To act on arbitrary length messages, blockciphers are typically used in so-called *modes*

[2] We note that there are some controversies regarding the interpretation and practical relevance of provable security. We touch upon this in Chapter 10.

[3] In this single case, we do *not* mean confidentiality, but rather the ability of a user to control the spread of information about themselves.

[4] From this point, we merge the authenticity and integrity into a single goal, as one is almost never useful without the other. We refer to the joint goal interchangeably by authenticity or integrity.

[5] Which was proven perfectly secure by Shannon [Sha45].

of operation, such as the CTR mode or CBC mode [Dwo01], which use a blockcipher as a blackbox subroutine to construct more general encryption schemes.

The authenticity of messages has typically been achieved with help of Message Authentication Codes (MACs). A MAC produces a fixed-size, hard-to-predict authentication tag as a function of a secret key and a message. MACs were most often constructed as modes of operation of blockciphers (e.g., the CBC-based MACs [BKR00, Dwo16]), but also other primitives such as compression functions [BCK96] or possibly streamciphers [CW77, WC81].

Confidentiality and authenticity were also captured by separate security notions. Confidentiality was formalized in several notions, most notably the Left-or-Right, or Real-or-Random indistinguishability [BDJR97]; informally, both of them imply that an adversary who can ask for encryptions of messages of his/her choice cannot tell if the ciphertexts he/she is getting back truly encrypt his/her inputs. The security of MACs was captured by the notions of unforgeability and pseudo-random function [BKR00]; the former requires that an adversary which can ask for tags of arbitrary messages cannot produce a fresh, valid message-tag pair, while the latter is a stronger notion that requires that the given MAC "behaves" as a random function.

Unlike in the theoretical work, such a clear separation of confidentiality and authenticity seldom occurs in practice. In most conceivable applications, one will need to ensure confidentiality in conjunction with authenticity (and integrity). In the absence of a systematic treatment, users of cryptography had to resort to ad-hoc measures, such as inserting redundancy into plaintexts and encrypting them with *some* encryption scheme, to achieve the two said properties. These failed more often than not.

The best example of such a failure is perhaps the Wired Equivalent Privacy protocol (WEP) used by the 802.11 wireless network standard. Here, the Cyclic Redundancy Check (a linear function of the plaintext) had been appended to the plaintext, and the resulting string was encrypted with the streamcipher RC4. The integrity protection of this construct was completely void, as demonstrated by Borisov, Goldberg and Wagner [BGW01]. Another such failure was the CBCC scheme, which appended to a message the XOR-sum of all its blocks, and applied the CBC mode to the result. This one was shown to be flawed by Menezes, van Oorschot, and Vanstone [MvOV96].

Even combining a secure encryption scheme with a secure MAC was not sure to yield a secure result. Canvel, Hiltgen, Vaudenay and Vuagnoux showed that an unfortunate interaction between a MAC and the padding used for the CBC mode allowed an adversary to compromise the confidentiality of plaintext in practical attacks on SSL and IPSec [Vau02, CHVV03].

Authenticated encryption. The unsatisfied need for a tool that would simultaneously provide both confidentiality and authenticity was picked up by the research community in 2000; Katz and Yung [KY00] and independently Bellare with Rogaway [BR00] stated the first formal security notions for what is today called *authenticated encryption* (AE). In that same year, Bellare and Namprempre formally analysed the security of *generic composition*, showing that given a secure probabilistic encryption scheme and a secure MAC, the Encrypt-then-MAC construction will yield a secure AE scheme.

In 2001, Jutla proposed IACBC and IAPM [Jut01], the first two provably secure dedicated AE schemes. In the same year, Rogaway, Bellare, Black and Krovetz designed OCB [RBBK01], one of the most popular dedicated AE schemes. In 2002, Rogaway put forward the syntax and security notion for nonce-based authenticated encryption with associated data, an additional input to the encryption algorithm that should be authenticated along with the message, but not encrypted [Rog02]. This syntax and security notion have become the most popular design target so far. Several other dedicated AE schemes followed soon after, most notably the Counter with CBCMAC (CCM) [WHF03b] and Galois/Counter Mode (GCM) [MV04]. All of the early AE schemes of importance were designed as blockcipher modes of operation.

Simultaneously with this burst of research activity, authenticated encryption also enjoyed a quick deployment in real-world applications. CCM appears in IEEE 802.11i, IPsec ESP and IKEv2, while GCM appears in NIST SP 800-38D. ISO/IEC 19772:2009 defines six AE schemes (five dedicated AE designs and one generic composition method).

However, soon after the first wave of dedicated AE schemes, a series of new problems with (some of) those schemes and with the applications of symmetric cryptography in general were discovered. Complaints about the security and/or applicability of CCM [RW03] and GCM [Jou06, Fer05, Saa12, IOM12] were raised and a collection of failures of symmetric cryptography was documented by Bernstein [Ber14b]. Rogaway and Shrimpton pointed out that most of the existing nonce-based AE schemes completely collapse if nonces get *misused* (i.e., repeated) and defined nonce-misuse resistant security as a solution for this [RS06b].

CAESAR competition. All these shortcomings indicated the need for further research in the field of authenticated encryption. To both boost the research, and to provide solutions for these problems, the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) was launched in 2013. It's official goal was to "identify a portfolio of authenticated ciphers that (1) offer advantages over AES-GCM and (2) are suitable for widespread adoption" [Ber14a]. The final portfolio was supposed to be identified in three rounds, each taking about a year.

The candidates submitted to CAESAR were required to comply with the specified syntax,[6] to ensure authenticity of all non-key inputs, and to protect confidentiality of the plaintext and so-called "secret message number." [Ber14a]. Each candidate was also required to clearly indicate the amount of data that can be securely processed with a single secret key, and whether the candidate

1. required that the nonce ("public message number" in CAESAR) never repeats, or

[6]This was essentially an extension of the syntax for nonce-based AE by Rogaway, optionally augmented by a "secret message number". It must be noted that the addition of the secret message numbers was not received very well [NRS13].

2. is nonce-misuse resistant in the sense defined by Rogaway and Shrimpton, or

3. "provides some intermediate level of robustness against message-number reuse, in which case [the candidate] must specify what that level of robustness is."

The boost in research activity was immediate: 57 candidates were submitted to the first round of CAESAR. Out of these, 9 candidates were withdrawn by their designers during the first round, mostly due to the discovery of serious flaws [Viz16]. The remaining 48 designs showed a great diversity of low-level primitives, construction paradigms and targeted security notions. There were:

- 21 blockcipher-based schemes,
- 3 designs based on (dedicated) tweakable blockciphers,
- 7 designs based on previously existing streamciphers,
- 9 "Sponge"-based schemes,
- 3 modes of cryptographic permutations (not based on the Sponge),
- a single candidate based on a keyed compression function,
- 3 candidates based on a dedicated primitive that did not fall into any of the, previous categories
- a single candidate not based on any symmetric key primitive [Viz16].

While the prevailing construction type were by far blockcipher modes of operation, we see that a good number of other constructions based on different primitives appeared as well. Among these, the most popular were the schemes inspired by the *Sponge* construction. The Sponge is a mode of operation for a keyless cryptographic permutation, originally designed for cryptographic hashing.

The most frequently targeted security notion among the 48 candidates was the (basic) security of nonce-based AE defined by Rogaway [Rog02], and 5 candidates targeted the nonce-misuse resistant security [RS06b]. Another security notion that was very popular among the 1st round CAESAR candidates was the then-newly-defined notion of *online misuse resistant AE* (OAE), targeted by 4 submissions. Security notions prompted by the discussions around the start of CAESAR competition were the Robust AE security [HKR15] and the security of AE under the release of unverified plaintext (RUP) [ABL+14a]. Most 1st round candidates were supported by proofs in (in the sense of provable security), but there were also those who only made claims accompanied with direct cyptanalysis.

In July 2015, 29 candidates advanced to the 2nd round of CAESAR. All candidates were allowed to introduce updates, two submissions were merged into a single one based on the suggestion of the committee (CLOC and SILC), and two candidates merged into a single submission at the end of the second round (and formed COLM). In March 2018, 7 finalists were announced.

Outline of the thesis. The work that resulted in this dissertation started together with the CAESAR competition, and finished shortly before the CAESAR finalists were announced; the author of the thesis was thus fortunate to participate in this second big wave of research in authenticated encryption—probably even the bigger of the two. Thanks to this, the author was able to work on several exciting topics which are not all closely related to one another, but which can all be labelled as topics in (or related to) provable security. As a result, the material contained in this thesis is quite diverse. The covered results are therefore presented in three parts, each of them dedicated to a different aspect of research on AE.

In Part I, we discuss *construction of AE schemes*. In Part II we switch from designing schemes to definitional work and propose new *security models*. In Part III, we depart further away from provable security and focus on *cryptanalysis*.

We introduce the notation and the general security model used in the thesis in Chapter 2. We also cover some low-level symmetric key primitives and the pre-CAESAR notions of AE security in the same chapter.

Part I. The first three chapters in this part are dedicated to compression function-based authenticated encryption. In Chapter 3, we introduce Offset Merkle-Damgård (OMD), the first dedicated AE scheme based on a keyed compression function, and a 2^{nd} round CAESAR candidate. We propose two misuse-resistant variants of OMD in Chapter 4. In Chapter 5, we introduce pure OMD, which folds the processing of associated data into the main encryption-core of the original OMD scheme. In these chapters, we give descriptions of the proposed schemes, and prove their security. For OMD and pure OMD, we discuss their performance, as indicated by results from experimental measurements. We also briefly consider OMD from the perspective of the CAESAR competition.

We then turn to sponge-based AE schemes in Chapter 6, and propose algorithmic modifications to the Sponge and Duplex constructions that allow to greatly increase the throughput of the said constructions. We prove that the proposed modifications do not impact security, and demonstrate how this is beneficial for sponge-based AE.

Part II. In Chapter 7, we critically examine the security notion of Online AE [FFL12] that claims to imply nonce misuse resistance for AE schemes with *online* encryption. We point out several flaws and shortcomings due to which, in our opinion, the notion does not deliver on the intuition that it promises to capture. We propose an alternative, a security definition called OAE2, which comes as close to nonce misuse resistance as it is possible for online AE schemes. We demonstrate the feasibility of OAE2 security by constructing a secure instance.

We keep to the definitional work in Chapter 8. We consider a new type of misuse: varying the ciphertext expansion of a nonce-based AE scheme with a single key. We investigate implications of this misuse on security of nonce-based AE schemes and present plausible attacks which apply to many existing schemes, even if they apply some heuristic countermeasures. We then proceed to formalizing the security of nonce-based AE with variable ciphertext expansion, and establish the relations between our newly proposed notions and the previously existing ones. We demonstrate that our notion of security is achievable by presenting a secure scheme.

Part III. We attempt to complement the results of provable security in Chapter 9 by making a survey of actual security of all 3$^{\text{rd}}$ round CAESAR candidates, CCM and GCM to nonce-misuse and attacks with high data complexity. In the process, we describe a few new attacks. Our result is an overview that compares attacks in terms of their impact, complexity, and type of adversarial powers. It provides fine-grained information on "robustness" of CAESAR candidates beyond what is indicated by their claims.

Finally, we conclude in Chapter 10 and state open problems.

Personal Bibliography

The papers published during the making of this thesis are listed below. Publications whose contents are included in this dissertation are in bold.

[1] **Serge Vaudenay and Damian Vizár. Under pressure: Security of CAESAR candidates beyond their guarantees. Cryptology ePrint Archive, Report 2017/1147, 2017. `https://eprint.iacr.org/2017/1147`.**

[2] Guillaume Endignoux and Damian Vizár. Linking online misuse-resistant authenticated encryption and blockwise attack models. *IACR Trans. Symmetric Cryptol.*, 2016(2):125–144, 2016.

[3] **Reza Reyhanitabar, Serge Vaudenay, and Damian Vizár. Authenticated encryption with variable stretch. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 396–425, 2016.**

[4] Tetsu Iwata, Bart Mennink and Damian Vizár. CENC is optimally secure. Cryptology ePrint Archive, Report 2016/1087, 2016. `https://eprint.iacr.org/2016/1087`

[5] Damian Vizár. Ciphertext forgery on HANUMAN. Cryptology ePrint Archive, Report 2016/697. IACR,2016. `https://eprint.iacr.org/2016/697`

[6] Damian Vizár. The State of the authenticated encryption. *Tatra Mountains Mathematical Publications*, 67(1):167–190, 2016.

[7] **Bart Mennink, Reza Reyhanitabar, and Damian Vizár. Security of full-state keyed sponge and duplex: applications to authenticated encryption. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 465–489. Springer, 2015.**

[8] **Viet Tung Hoang, Reza Reyhanitabar, Phillip Rogaway and Damian Vizár. Online authenticated-encryption and its nonce-reuse misuse-resistance. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 493–517. Springer, 2015.**

[9] Serge Vaudenay and Damian Vizár. Cryptanalysis of chosen symmetric homomorphic schemes. *Studia Scientiarum Mathematicarum Hungarica*, 52(2):288–306, 2015.

[10] **Reza Reyhanitabar, Serge Vaudenay and Damian Vizár. Boosting OMD for almost free authentication of associated data. In Gregor Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 411–427. Springer, 2015.**

[11] **Reza Reyhanitabar, Serge Vaudenay and Damian Vizár. Misuse-resistant variants of the OMD authenticated encryption mode. In Sherman S. M. Chow, Joseph K. Liu, Lucas Chi Kwong Hui, and Siu-Ming Yiu, editors, *Provable Security - 8th International Conference, ProvSec 2014, Hong Kong, China, October 9-10, 2014. Proceedings*, volume 8782 of *Lecture Notes in Computer Science*, pages 55–70. Springer, 2014.**

[12] **Simon Cogliani, Diana-Stefania Maimut, David Naccache, Rodrigo Portella do Canto, Reza Reyhanitabar, Serge Vaudenay and Damian Vizár. OMD: A compression function mode of operation for authenticated encryption. In Antoine Joux and Amr M. Youssef, editors, *Selected Areas***

***in Cryptography - SAC 2014 - 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers*, volume 8781 of *Lecture Notes in Computer Science*, pages 112–128. Springer, 2014.**

The non-included publications. In the paper that discusses online misuse-resistant authenticated encryption (OAE) [FFL12] and blockwise attack models [FJMV03], we attempt to determine the relations between these two security notions [2]. Because the former notion works with deterministic schemes, and the latter with randomized ones, they are not directly comparable. We therefore recast the blockwise security definitions to work with deterministic AE schemes, and establish that the resulting notion is almost equivalent with OAE security, except that the blockwise notions do not capture the pseudorandomness of the authentication tag, but OAE does.

In the note on the encryption scheme CENC [Iwa06], we point out a connection between two results that seems not to have been captured in the literature before [4]. More precisely, we show how Patarin's results on the PRF-security of the XOR of random permutations [Pat08b, Pat10] imply the optimal security of CENC (whose original analysis proved suboptimal beyond-birthday-bound security).

In the short note about the ciphertext forgery on HANUMAN [5], we show that the AE scheme HANUMAN, which was a part of the 2$^{\text{nd}}$ round CAESAR submission PRIMATEs [ABB$^+$14b], contains a flaw that makes the domain separation between queries with complete and incomplete associated data imperfect. We demonstrate how this flaw can be used to mount forgeries with constant complexity.

The work that discusses the state of authenticated encryption in year 2016 [6] was an accompanying paper for the invited talk at the Central European Conference on Cryptology 2016. It briefly lists the milestones and events that preceded the CAESAR competition and gives a summary of the first and second rounds of CAESAR. It also gives a concise description of the CAESAR candidate OMD.

Chapter 2

Preliminaries

This chapter introduces the notations and definitions used in the rest of this thesis. In Section 2.1, we introduce basic concepts and notations used throughout this dissertation. In Section 2.2, we briefly discuss the formal treatment we apply analysing the security of cryptographic constructions. In Section 2.3 we formally define several cryptographic primitives and their security. In Section 2.4 we formally define syntax of schemes for authenticated encryption, and define several notions of security for such schemes.

2.1 Notations

Sampling, sets and integers. We let $a \leftarrow\$ \mathcal{S}$ denote sampling a random variable $\mathcal{S}$ and storing the result in the variable a. If $\mathcal{S}$ is a finite set, we denote by $a \leftarrow\$ \mathcal{S}$ sampling an element of $\mathcal{S}$ uniformly at random. For a finite set $\mathcal{S}$, we let $|\mathcal{S}|$ denote its cardinality.

We let $\mathbb{N}$ denote the set of all natural numbers $\{0, 1, 2, \ldots\}$ and $\mathbb{N}^+$ the set of all positive natural numbers $\mathbb{N}\backslash\{0\}$. For any two pairs of integers $(i, j), (i', j') \in \mathbb{N}$, we say that $(i', j') < (i, j)$ if either $i' < i$, or if $i' = i$ and $j' < j$. We say that $(i', j') \leq (i, j)$ if $(i', j') < (i, j)$ or if $(i', j') = (i, j)$. In other words, we use lexicographic ordering to determine ordering of integer-tuples.

Binary strings. All strings are binary strings, i.e. strings over the alphabet $\{0, 1\}$. We let $|X|$ denote the length of a string X. We let ε denote the empty string of length 0. We let $\{0, 1\}^*$ denote the set of all strings of arbitrary finite lengths (s.t. $\varepsilon \in \{0, 1\}^*$), we let $\{0, 1\}^\infty$ denote the set of all infinite strings and we let $\{0, 1\}^n$ denote the set of all strings of length n for a non-negative integer n. We further let $\{0, 1\}^{\leq n}$ denote $\bigcup_{i=0}^{n} \{0, 1\}^i$, the set of all strings of length equal or smaller than n, and we identify $\{0, 1\}^{<n}$ with $\{0, 1\}^{\leq n-1}$ for $n \geq 2$. We also let $\{0, 1\}^{\geq n}$ denote the set of all strings of at least n bits $\bigcup_{i \geq n}^{\infty} \{0, 1\}^i$.

Concatenation and substrings. We denote the concatenation of two strings X and Y by $X\|Y$, or by XY if the meaning of the notation is clear from the context. For an m-bit string $X = X[m-1]\cdots X[0]$ we let $X[i\cdots j] = X[i]\cdots X[j]$ denote a substring of X, for $m-1 \geq i \geq j \geq 0$; by convention we let $X[i\cdots j] = \varepsilon$ if $i < 0$ and $X[i\cdots j] = X[i\cdots 0]$ if $j < 0$. We further let $\mathsf{left}_\ell(X) = X[(m-1)\ldots(m-\ell)]$ denote the ℓ leftmost bits of X and $\mathsf{right}_r(X) = X[(r-1)\ldots 0]$ the r rightmost bits of X, such that $X = \mathsf{left}_\chi(X)\,\|\mathsf{right}_{|X|-\chi}(X)$ for any $0 \leq \chi \leq |X|$.

Length of blockwise common prefix. Given two strings X, Y, let

$$\mathsf{llcp}_b(X,Y) = \max_{i\geq 0}\{i \;:\; \mathsf{left}_{i\cdot b}(X) = \mathsf{left}_{i\cdot b}(Y)\}$$

denote the **l**ength of the **l**ongest **c**ommon **p**refix between X and Y in b-bit blocks. For a string X and a non-empty set of strings $\{Y_1,\ldots,Y_n\}$ let

$$\mathsf{llcp}_b(X;Y_1,\ldots,Y_n) = \max\{\mathsf{llcp}_b(X,Y_1),\ldots,\mathsf{llcp}_b(X,Y_n)\}\,.$$

Encodings, number of trailing zeros and partitioning of strings. For a non-negative integer $i \in \mathbb{N}$ let $\langle i\rangle_m$ denote the binary representation of i as an m-bit string. For a string $X = X[m-1]\cdots X[0]$, let $\mathsf{int}(X) = \sum_{i=0}^{m-1} X[i]2^i$ denote the non-negative integer represented by X.

Let $\mathtt{ntz}(i)$ denote the number of trailing zeros (i.e., the number of rightmost bits that are zero) in the binary representation of a positive integer i. E.g. $\mathtt{ntz}(7) = 0$ and $\mathtt{ntz}(12) = 2$. Let b^n denote the string obtained by concatenating n copies of a bit $b \in \{0,1\}$.

For $X \in \{0,1\}^*$ let $X_1\|X_2\cdots\|X_m \xleftarrow{b} X$ denote partitioning X into blocks such that $|X_i| = b$ for $1 \leq i \leq m-1$ and $|X_m| \leq b$; let $m = |X|_b$ denote length of X in b-bit blocks (i.e., $|X|_b = \lceil |X|/b\rceil$).

XOR and binary shifts. For two strings X and Y with $|X| = |Y|$, we let $X \,\&\, Y$ denote the result of the bitwise and operation applied to X and Y.

For two strings X and Y, with $|X| \leq |Y|$, let the notations $X \oplus Y$ and $Y \oplus X$ both denote the bitwise xor of X and $\mathsf{left}_{|X|}(Y)$. Clearly, if X and Y have the same length then $X \oplus Y$ matches the usual bitwise xor. For any string X, this implies $X \oplus \varepsilon = \varepsilon \oplus X = \varepsilon$.

For a string X, let $X \ll n = \mathsf{right}_{|X|-n}(X)\,\|0^n$ denote the left-shift operation, where the n leftmost bits are discarded and the n vacated rightmost bits are set to 0. We let $X \gg n = 0^n\|\mathsf{left}_{|X|-n}(X)$ denote the (unsigned) right-shift operation where the n rightmost bits are discarded and the n vacated leftmost bits are set to 0. We let $X \gg_s n = (\mathsf{left}_1(X))^n\,\|\mathsf{left}_{|X|-n}(X)$ denote the *signed* right-shift operation where the n rightmost bits are discarded and the n vacated left bits are filled with the original leftmost bit (which is considered as the sign bit); for example, $1001100 \gg_s 3 = 1111001$. If the leftmost bit of X is 0 then we have $X \gg_s n = X \gg n$.

Finite fields. For a positive integer n, let $\mathsf{GF}(2^n)$ denote the Galois Field with 2^n elements. An element α in $\mathsf{GF}(2^n)$ is represented as a formal polynomial $\alpha(X) = \alpha_{n-1}X^{n-1} + \cdots + \alpha_1 X + \alpha_0$ with binary coefficients.

We can assign an element $\alpha \in \mathsf{GF}(2^n)$ to an integer $i \in \{0, \ldots, 2^n - 1\}$ in the natural way. A natural assignment of an element α exists for any string $s \in \{0,1\}^n$. With an abuse of notation, we sometimes refer to the elements of $\mathsf{GF}(2^n)$ directly by strings in $\{0,1\}^n$, or integers in $\{0, \ldots, 2^n - 1\}$, if the context does not allow ambiguity.[1]

The addition "$\oplus$" and multiplication "$\cdot$" of two field elements in $\mathsf{GF}(2^n)$ are defined as usual. For the representation of $\mathsf{GF}(2^{256})$ we use the polynomial $P_{256}(X) = X^{256} + X^{10} + X^5 + X^2 + 1$, and for $\mathsf{GF}(2^{512})$ we use the polynomial $P_{512}(X) = X^{512} + X^8 + X^5 + X^2 + 1$ as the modulus used in the field multiplications.

We note that from the perspective of implementation, it is easy to multiply an arbitrary field element α by the element identified with 2 (i.e., X). For example, in $\mathsf{GF}(2^{256})$ using $P_{256}(X)$ the doubling[2] operation can be described as follows:

$$2 \cdot \alpha = \begin{cases} \alpha \ll 1 & \text{if } \mathsf{left}_1(\alpha) = 0 \\ (\alpha \ll 1) \oplus 0^{245}10000100101 & \text{if } \mathsf{left}_1(\alpha) = 1 \end{cases} \tag{2.1}$$

$$= (\alpha \ll 1) \oplus ((\alpha \gg_s 255) \,\&\, 0^{245}10000100101) \tag{2.2}$$

The error symbol. The special symbol $\perp$ signifies either an error or that the value of a variable or a function at some input is undefined.

Arrays and vectors. An array S stores associations (X, Y) between elements of a set of indices $X \in \mathcal{D}$ and a set of values $Y \in \mathcal{R}$. An array S can also be thought of as function $S : \mathcal{D} \to \mathcal{R} \cup \{\perp\}$ that can be dynamically modified. We assume all the standard operations over the arrays: efficient random read and write access, and adding or removing elements.

We denote by $S \leftarrow \text{array}(\mathcal{D})$ initializing S to an empty array with the set of indices $\mathcal{D}$. An empty array contains no association. Given an array S and some $X \in \mathcal{D}$ for which an association with some Y was stored in S, we let $S[X] = Y$. If no association was stored for X, we let $S[X] = \perp$ by convention. This means that for an empty array S, $S[X] = \perp$ for all $X \in \mathcal{D}$. Given an array S, we let $|S|$ denote the number of associations stored in S, i.e., the number of $X \in \mathcal{D}$ such that $S[X] \neq \perp$. We assume that the memory needed to store S is equal to $\gamma \cdot \log(|\mathcal{D}|) \cdot |\{X \in \mathcal{D} \mid S[X] \neq \perp\}|$ for some constant γ.

We conflate vectors with lists, i.e., we do not understand a vector as an element of a vector space in the algebraic sense, unless explicitly stated otherwise. Given a set $\mathcal{R}$, a

[1] We adopt this rather abusive notation because it is very common in the works on AE and tweakable blockciphers.

[2] We note that here doubling does not mean "adding an element to itself" but the multiplication by the $\mathsf{GF}(2^n)$-element that can be represented by the integer 2, i.e., the formal polynomial $\alpha(X) = X$.

vector $V \in \mathcal{R}^*$ is simply a tuple that contains a non-negative number of ordered elements of $\mathcal{R}$. We let $|V|$ denote the number of elements in the vector V. We let Λ denote the empty vector with $|\Lambda| = 0$ for any $\mathcal{R}$. For a vector V, we let $V[i]$ denote the i^{th} element of V for $1 \leq i \leq |V|$, and we let $V[i..j]$ denote the vector $(V[i], V[i+1], \ldots, V[j])$ for $1 \leq i < j \leq |V|$. We let $V[i \ldots i] = V[i]$ by convention. For $i \geq j$ we let $\Lambda = V[i..j]$ by convention. For a vector $V \in \mathcal{R}^*$ and an element $X \in \mathcal{R}$, we let $V \| X$ denote the vector $(V[1], \ldots, V[|V|], X)$.

Functions, permutations and injections. For a bijective function $\pi : \mathcal{S} \to \mathcal{S}$, we denote its inverse by $\pi^{-1} : \mathcal{S} \to \mathcal{S}$, i.e., for all $s \in S$ we have $\pi^{-1}(\pi(s)) = s$.

For a finite set $\mathcal{S}$, we let $\mathrm{Perm}(\mathcal{S})$ denote the set of all permutations of $\mathcal{S}$. For a positive n, we let $\mathrm{Perm}(2^n) = \mathrm{Perm}(\{0,1\}^n)$.

For a set $\mathcal{D}$ and a finite set $\mathcal{R}$, we let $\mathrm{Func}(\mathcal{D}, \mathcal{R})$ denote the set of all functions from $\mathcal{D}$ to $\mathcal{R}$, and if $\mathcal{D} = \mathcal{R}$, we simply use $\mathrm{Func}(\mathcal{D})$ instead of $\mathrm{Func}(\mathcal{D}, \mathcal{D})$. In particular, we let $\mathrm{Func}(\mathcal{D}, 2^n) = \mathrm{Func}(\mathcal{D}, \{0,1\}^n)$, and $\mathrm{Func}(2^m, 2^n) = \mathrm{Func}(\{0,1\}^m, \{0,1\}^n)$, and $\mathrm{Func}(2^n) = \mathrm{Func}(\{0,1\}^n)$ for positive integers m and n.

For two finite sets $\mathcal{D}$ and $\mathcal{R}$ with $|\mathcal{D}| \leq |\mathcal{R}|$, we let $\mathrm{Inj}(\mathcal{D}, \mathcal{R})$ denote the set of all injective functions from $\mathcal{D}$ to $\mathcal{R}$. In particular, we let $\mathrm{Inj}(2^m, 2^n) = \mathrm{Inj}(\{0,1\}^m, \{0,1\}^n)$ for positive integers m and n with $m \leq n$.

For a positive integer τ, we let $\mathrm{Inj}(\tau)$ denote the set of all τ-expanding injections, i.e., the set of functions $f : \{0,1\}^* \to \{0,1\}^*$ s.t. for all $M \in \{0,1\}^*$ we have $|f(M)| = |M| + \tau$ and for all $M \neq M' \in \{0,1\}^*$ we have $f(M) \neq f(M')$.

We additionally define so called "tweakable" version of the set of permutations and the set of functions. For a set $\mathcal{T}$ and a finite set $\mathcal{S}$, we let $\widetilde{\mathrm{Perm}}(\mathcal{T}, \mathcal{S})$ denote the set of all functions $f : \mathcal{T} \times \mathcal{S} \to \mathcal{S}$ such that for all $\mathsf{T} \in \mathcal{T}$, $f(\mathsf{T}, \cdot)$ is a permutation of $\mathcal{S}$. In particular, we let $\widetilde{\mathrm{Perm}}(\mathcal{T}, 2^n) = \widetilde{\mathrm{Perm}}(\mathcal{T}, \{0,1\}^n)$ for a positive integer n. For a $\widetilde{\pi} \in \widetilde{\mathrm{Perm}}(\mathcal{T}, \mathcal{D})$, we let $\widetilde{\pi}^{\mathsf{T}}(X) = \widetilde{\pi}(\mathsf{T}, X)$.

A set of functions from $\mathcal{D}$ to $\mathcal{R}$ tweaked by $\mathcal{T}$ is identical to $\mathrm{Func}(\mathcal{T} \times \mathcal{D}, \mathcal{R})$. We define the corresponding notation nevertheless, as it will be useful in Chapters 3, 4 and 5. For a set $\mathcal{T}$ and two positive integers m, n, we let $\widetilde{\mathrm{Func}}(\mathcal{T}, 2^m, 2^n)$ stand for $\mathrm{Func}(\mathcal{T} \times \{0,1\}^m, \{0,1\}^n)$, such that we treat the elements of the domain $\mathcal{T} \times \{0,1\}^m$ as tweak-message pairs. For an $\widetilde{f} \in \widetilde{\mathrm{Func}}(\mathcal{T}, 2^m, 2^n)$, we let $\widetilde{f}^{\mathsf{T}}(X) = \widetilde{f}(\mathsf{T}, X)$.

Note that for a $\widetilde{f} \in \widetilde{\mathrm{Func}}(\mathcal{T}, 2^m, 2^n)$, we can interpret $\widetilde{\pi} = (\widetilde{\pi}^{\mathsf{T}}(\cdot))_{\mathsf{T} \in \mathcal{T}}$ as a collection of functions indexed by T. This interpretation will be used in particular if the notation $\widetilde{\mathrm{Func}}(\mathcal{T}, 2^m, 2^n)$ appears.

Algorithms and oracles. We let $\mathscr{A}^{\mathcal{O}_1(\cdot), \ldots, \mathcal{O}_r(\cdot)}$ denote that an algorithm $\mathscr{A}$ has black-box access to algorithms $\mathcal{O}_1, \ldots, \mathcal{O}_r$ such that it can feed inputs to and observe corresponding outputs of each $\mathcal{O}_i$ for $i = 1, \ldots, r$. We call $\mathcal{O}_1, \ldots, \mathcal{O}_r$ the oracles of $\mathscr{A}$. When $\mathscr{A}$ runs $Y \leftarrow \mathcal{O}_i(X)$ with an input X, we say that $\mathscr{A}$ makes a query to $\mathcal{O}_i$, or that $\mathscr{A}$ queries $\mathcal{O}_i$ with X. We call Y response to $\mathscr{A}$'s query.

We let $\mathscr{A}^{\mathcal{O}_1(\cdot),\ldots,\mathcal{O}_r(\cdot)}(X) \Rightarrow a$ denote the event that an algorithm $\mathscr{A}$ that is fed an input X and that has black-box access to oracles $\mathcal{O}_1, \ldots, \mathcal{O}_r$ outputs the value a.

We let $S \leftarrow\!\$\ \mathcal{S} : \mathscr{A}^{\mathcal{O}_1(\cdot),\ldots,\mathcal{O}_r(\cdot)}()$ denote sampling the variable S according to $\mathcal{S}$, and then running $\mathscr{A}$ with its oracles, such that the variable S is implicitly accessible by the oracles, and may or may not be given to $\mathscr{A}$.

We note that this simple notation will only be used if $\mathscr{A}$'s oracles are simple enough to leave no room for ambiguity. For more complex scenarios, the interaction between an adversary and a set of oracles will be treated explicitly using the game-playing framework defined in Section 2.2.

2.2 Security Framework

In this section, we briefly discuss the approach we adopt for the formal treatment of security analyses in this thesis.

Model of computation. Unless stated otherwise, all algorithms are understood to be executed in the same, fixed RAM model of computation, and represented using the same algorithm encoding. By convention, the time complexity of an algorithm A is the worst-case running time of A plus the size of the description of A.

Concrete security. We apply the concrete security-treatment [BDJR97] when defining and analysing security of cryptographic primitives, as opposed to the asymptotic security treatment that is usual for public-key cryptography. We analyse the security of cryptographic constructions against adversaries whose resources (time complexity, data complexity etc.) are parameterized, and the upper bounds on the adversarial advantage are expressed as concrete functions of these parameters. Loosely speaking, adversarial advantage is a measure of how likely it is that an adversary is able to "break" a scheme, the meaning of "breaking" being scheme-dependent.

The reason for applying the concrete security treatment is twofold. Primarily, there is only a (small) finite number of instances of virtually any symmetric key construction. In particular, the security level guaranteed by these instances cannot be scaled at whim, which makes asymptotic security analysis meaningless. The concrete security analysis also gives a much more tangible assessment of a construction's actual level of resilience to attacks.

Indistinguishability. When formally defining the meaning of security for a cryptographic primitive, it is often convenient to do so through the black-box *indistinguishability* of a construction Π (that realizes the given primitive) from an idealized version of the primitive, i.e., a reference object with the same input/output interface.

The idealized version of the primitive (call it $\bar{\Pi}$) usually possesses a perfect version of the security properties that we would like Π to have. We then evaluate Π's security through an experiment, where we give an adversary $\mathscr{A}$ oracle access to either Π keyed with a random K (sampled from the corresponding key space $\mathcal{K}$), or to $\bar{\Pi}$ selected from the distribution $\bar{\mathbf{\Pi}}$ associated to the idealized object. Depending on the primitive, the adversary has access to one or more oracles, and its goal is to distinguish whether it is interacting with Π or $\bar{\Pi}$. That is, $\mathscr{A}$ outputs a single bit at the end of its interaction with the oracles. We (typically) measure the (in)security of Π as an expression of the form

$$\Pr[K \leftarrow\!\$\ \mathcal{K} : \mathscr{A}^{\Pi_K} \Rightarrow 1] - \Pr[\bar{\Pi} \leftarrow\!\$\ \bar{\mathbf{\Pi}} : \mathscr{A}^{\bar{\Pi}} \Rightarrow 1]$$

(where we are abusing the notation slightly), called the *adversarial advantage.*

The harder it is for an adversary to distinguish the actual keyed construction Π from the idealized version $\bar{\Pi}$, the more faithfully Π approximates the ideal reference $\bar{\Pi}$, and the smaller the advantage. Informally speaking, if the adversarial advantage is low for all possible adversaries whose computational resources (running time, memory complexity, the data complexity of the oracle queries) are limited by some reasonable upper bound, we say that Π is secure.

We note that if the idealized reference object $\bar{\Pi}$ is chosen appropriately, any valid attack that can be mounted with black-box access to Π can be turned into a black-box distinguisher. The absence of efficient distinguishers then necessarily implies the absence of any other efficient attacks. The tricky part of this approach is the choice of an appropriate reference object.

The concept of indistinguishability will be used to define the security of most of the cryptographic primitives in this thesis.

Adversarial powers. When analysing security of certain primitives, we consider adversaries who are given privileges of varying potency.

For primitives such as blockciphers, encryption-only schemes or authenticated encryption schemes, the adversary $\mathscr{A}$ may only be allowed to query the encryption algorithm. In that case, we say that $\mathscr{A}$ mounts a *chosen plaintext attack* (CPA). If the adversary can query both the encryption and the decryption algorithm, we say that it mounts a *chosen ciphertext attack* (CCA).

When choosing a set of “reasonable” adversaries for defining security of a primitive, we distinguish two cases. If we only consider adversaries that run in time that is limited by some constant t, we talk about *computational* security, as the primitive in question is only expected to resist attackers with a limited amount of computational power. Unless indicated otherwise, an adversary should always be assumed to be computationally restricted.

If we allow adversaries to run in unrestricted (but finite) time, we talk about *information-theoretic* security. We refer to such computationally unrestricted adversaries as information-theoretic adversaries.

We note that when considering an information-theoretic adversary, it can be assumed to be deterministic without loss of generality (for a simple proof see the paper by Chen and Steinberger [CS14]).

Game-playing framework. We use the game-playing methodology [BR06] to define security of cryptographic primitives and to conduct provable security analysis.

A game is an algorithm that consists of a collection of procedures; it describes a security experiment. An adversary $\mathscr{A}$ is a possibly randomized algorithm, that is left to interact with one or more procedures of the game called oracles. A game is executed in three phases.

First, a procedure called *initialize* is executed to set up the game and produce an initial input for $\mathscr{A}$. Then $\mathscr{A}$ is run with this input and left to interact with the oracles defined by the game. At the end of this interaction, $\mathscr{A}$ produces an output value. This value is passed to a procedure called *finalize* which then produces the final output of the game.

Overloading the notation, we let $\mathscr{A}^G \Rightarrow a$ denote the event that the finalize procedure outputs the value a when running a game G with an adversary $\mathscr{A}$.

If the initialization procedure is a dummy algorithm with no instructions, we may omit it completely. Similarly, if the finalization procedure is not specified, then we default to a dummy procedure that simply forwards the output of the adversary.

To describe the games, we will often use pseudo code, similar to that proposed by Bellare and Rogaway [BR06]. If the game is simple enough so that there is no risk of ambiguity, we dispense with the pseudo code and describe the game using simple "mathematical" notation. An example of the latter are the security definitions in Section 2.3.

The fundamental lemma of game-playing. We can apply the fundamental lemma of game-playing when we wish to upper-bound the quantity

$$\Pr[\mathscr{A}^{G_1} \Rightarrow a] - \Pr[\mathscr{A}^{G_2} \Rightarrow a]$$

for an adversary $\mathscr{A}$ and two games G_1 and G_2 that are *identical-until-bad*.

We call two games G_1 and G_2 identical-until-bad if the two games are syntactically identical except when a boolean flag bad (which exists in both of them) is set to true. More precisely, G_1 and G_2 are identical-until-bad if their code is the same except inside **if**-blocks that *always* set the flag bad to true if entered. The fundamental lemma says that the advantage that an adversary can obtain in distinguishing a pair of identical-until-bad games is at most the probability that its execution sets bad in one of the games (either game will do).

Lemma 2.1 (Fundamental lemma of game-playing [BR06])**.** *Let G_1 and G_2 be identical-until-bad games, let a be any value, and let $\mathscr{A}$ be an adversary. Then we have*

$$\Pr[\mathscr{A}^{G_1} \Rightarrow a] - \Pr[\mathscr{A}^{G_2} \Rightarrow a] \leq \Pr[\mathscr{A}^{G_i} \text{ sets } \mathsf{bad}],$$

where $i \in \{1, 2\}$ and where $\mathscr{A}^{G_i}$sets bad *denotes the event that the flag* bad *is set to* true *when the procedure finalize of G_i is called.*

Coefficient-H technique When analysing the indistinguishability of two games X and Y such that the oracles of X have identical interfaces with the oracles of Y, we can make use of the Patarin's Coefficient-H technique [Pat08a]; more precisely, a revisited formulation of it by Chen and Steinberger [CS14].

Consider a fixed information-theoretic adversary $\mathscr{A}$, which is w.l.o.g. *deterministic*, and whose goal is to distinguish the two games X and Y. Then we typically seek an upper bound for the quantity

$$\Delta_{\mathscr{A}}(X;Y) = \left|\Pr\left[\mathscr{A}^X \Rightarrow 1\right] - \Pr\left[\mathscr{A}^Y \Rightarrow 1\right]\right|.$$

$\mathscr{A}$'s interaction with any of the two games X or Y can be summarized in a *transcript* τ. A transcript is an ordered collection of $\mathscr{A}$'s oracle queries and the corresponding responses $\mathscr{A}$ received back form the oracles.

Given the game X and the adversary $\mathscr{A}$, we can define the random variable $D_X^{\mathscr{A}}$ as the transcript produced when we run $\mathscr{A}^X$. We define the random variable $D_Y^{\mathscr{A}}$ similarly. The domain of both $D_X^{\mathscr{A}}$ and $D_Y^{\mathscr{A}}$ is the set of all possible transcripts that $\mathscr{A}$ can produce when interacting with either X or Y. The probability distribution of $D_X^{\mathscr{A}}$ is defined over the random coins of X (recall that $\mathscr{A}$ is deterministic), and similarly, the distribution of $D_Y^{\mathscr{A}}$ is defined over the coins of Y.

A transcript τ is called Y-*attainable* if $\Pr\left[D_Y^{\mathscr{A}} = \tau\right] > 0$, meaning that it can occur during the interaction of $\mathscr{A}$ with Y. The Coefficient-H technique states the following (for the proof of which we refer to Chen and Steinberger [CS14]).

Lemma 2.2 (Coefficient-H Technique [Pat08a, CS14]). *Consider a fixed deterministic adversary $\mathscr{A}$. Let $\mathcal{T} = \mathcal{T}_{\text{good}} \cup \mathcal{T}_{\text{bad}}$ be a partition of the set of all Y-attainable transcripts $\mathcal{T}$. If there exists an ε such that for all $\tau \in \mathcal{T}_{\text{good}}$,*

$$\frac{\Pr\left[D_X^{\mathscr{A}} = \tau\right]}{\Pr\left[D_Y^{\mathscr{A}} = \tau\right]} \geq 1 - \varepsilon,$$

then $\Delta_{\mathscr{A}}(X;Y) \leq \varepsilon + \Pr\left[D_Y^{\mathscr{A}} \in \mathcal{T}_{\text{bad}}\right]$.

The two partitions of $\mathcal{T}$ are labelled as $\mathcal{T}_{\text{good}}$ and $\mathcal{T}_{\text{bad}}$ to increase the intuitiveness of the proof. The transcripts in $\mathcal{T}_{\text{good}}$ are "good" in the sense that they give us a high value of $\Pr[D_X = \tau]/\Pr[D_Y = \tau]$ and thus small ε while the "bad" transcripts from $\mathcal{T}_{\text{bad}}$ fail to do so.

We further note that in place of the game X we usually have a cryptographic construction, and in place of Y its idealized reference. Therefore, we often refer to the former as the "real world" and to the latter as the "ideal world".

2.3 Low-Level Primitives

In this section, we define the syntax and security of low-level cryptographic primitives that will be used in the remaining chapters.

Blockciphers. Blockciphers are one the most common building blocks of other symmetric primitives. A blockcipher allows to encrypt bitstrings of fixed size, so called *blocks*, into ciphertexts of the same size, in an invertible way.

A blockcipher is a pair of deterministic algorithms $\mathsf{B} : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ and $\mathsf{B}^{-1} : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ such that B maps a secret key $K \in \mathcal{K}$ and a plaintext block $X \in \{0,1\}^n$ to a ciphertext block $\mathsf{B}(K,X) \in \{0,1\}^n$, and such that for every $K \in \mathcal{K}$ and $X \in \{0,1\}^n$ we have

$$\mathsf{B}^{-1}(K, \mathsf{B}(K,X)) = X.$$

That is, B must be efficiently invertible.[3]

We let $\mathsf{B}_K(X) = \mathsf{B}(K,X)$ and similarly $\mathsf{B}_K^{-1}(X) = \mathsf{B}^{-1}(K,X)$. We call n, a positive integer, the *blocksize* of B and $\mathcal{K}$, a finite set, the *key space* of B. We note that we typically have $\mathcal{K} = \{0,1\}^k$ for some positive integer k. We further note that the required invertibility of B implies that for every $K \in \mathcal{K}$, both $\mathsf{B}(K,\cdot)$ and $\mathsf{B}^{-1}(K,\cdot)$ are necessarily permutations of $\{0,1\}^n$. We use B to denote both the blockcipher and its encryption algorithm.

We quantify the security of a blockcipher through the notion of computational indistinguishability from a random permutation (RP). The intuition behind this notion is the following. When keyed with a secret key, the blockcipher (with a blocksize of n bits) must necessarily be a permutation of $\{0,1\}^n$. An ideal blockcipher should behave completely "random" beyond this constraint. To an adversary who only sees the keyed blockcipher as a black box, the best possible blockcipher will look identical to a random permutation of $\{0,1\}^n$.

Definition 2.3 ((S)PRP security)**.** *Given a blockcipher* $\mathsf{B} : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ *and an adversary* $\mathscr{A}$ *that has black-box access to* B*, we define the advantage of* $\mathscr{A}$ *in breaking the security of* B *in a chosen plaintext attack as*

$$\mathbf{Adv}_{\mathsf{B}}^{\mathbf{prp}}(\mathscr{A}) = \Pr\left[K \leftarrow\$\, \mathcal{K} : \mathscr{A}^{\mathsf{B}_K(\cdot)} \Rightarrow 1\right] - \Pr\left[\pi \leftarrow\$\, \mathrm{Perm}(2^n) : \mathscr{A}^{\pi(\cdot)} \Rightarrow 1\right].$$

If the advantage $\mathbf{Adv}_{\mathsf{B}}^{\mathbf{prp}}(\mathscr{A}) \le \epsilon$ *for every adversary* $\mathscr{A}$ *with running time and query complexity bounded by* t *and* q *respectively, we say that* B *is a* (ϵ, t, q)*-secure* pseudorandom permutation (PRP)*.*

For an adversary $\mathscr{A}$ *that has black-box access to* B *and* B^{-1}*, we define the advantage of* $\mathscr{A}$ *in breaking the security of* B *in a chosen ciphertext attack as*

$$\mathbf{Adv}_{\mathsf{B}}^{\mathbf{sprp}}(\mathscr{A}) = \Pr\left[K \leftarrow\$\, \mathcal{K} : \mathscr{A}^{\mathsf{B}_K(\cdot),\mathsf{B}_K^{-1}(\cdot)} \Rightarrow 1\right] - \Pr\left[\pi \leftarrow\$\, \mathrm{Perm}(2^n) : \mathscr{A}^{\pi(\cdot),\pi^{-1}(\cdot)} \Rightarrow 1\right].$$

If the advantage $\mathbf{Adv}_{\mathsf{B}}^{\mathbf{sprp}}(\mathscr{A}) \le \epsilon$ *for every adversary* $\mathscr{A}$ *with running time bounded by* t*, and encryption and decryption query complexity bounded by* q_e *and* q_d *respectively, we say that* B *is a* (ϵ, t, q_e, q_d)*-secure* strong pseudorandom permutation (SPRP)*.*

[3] We note that this notation is slightly abusive as B^{-1} is not an inverse of B in the strictest sense.

Remark 1. *We note that we may simply measure the adversarial resources by $q = q_e + q_d$ for SPRP security.*

Tweakable blockciphers Tweakable blockciphers are an extension of blockciphers that adds an extra input: the tweak [LRW02]. Changing the tweak with the same key in a tweakable blockcipher should mimic the effects of re-keying it with independent keys.

A tweakable blockcipher is a pair of efficient deterministic algorithms $\widetilde{\mathsf{B}} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$ and $\widetilde{\mathsf{B}}^{-1} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$, such that $\widetilde{\mathsf{B}}$ maps a secret key $K \in \mathcal{K}$, a tweak $T \in \mathcal{T}$ and a plaintext block $X \in \{0,1\}^n$ to a ciphertext block $\widetilde{\mathsf{B}}(K,T,X) \in \{0,1\}^n$, and such that for every $K \in \mathcal{K}$, $T \in \mathcal{T}$ and $X \in \{0,1\}^n$ we have

$$\widetilde{\mathsf{B}}^{-1}(K,T,\mathsf{B}(K,T,X)) = X.$$

That is, $\widetilde{\mathsf{B}}$ must be efficiently invertible.

We let $\widetilde{\mathsf{B}}_K^T(X) = \widetilde{\mathsf{B}}_K(T,X) = \mathsf{B}(K,T,X)$ and similarly $\widetilde{\mathsf{B}}_K^{-1}{}^T(X) = \mathsf{B}_K^{-1}(T,X) = \mathsf{B}^{-1}(K,T,X)$. We call n, a positive integer, the *blocksize* of $\widetilde{\mathsf{B}}$, we call $\mathcal{K}$, a finite set, the *key space* of $\widetilde{\mathsf{B}}$, and we call $\mathcal{T}$, a finite set, the *tweak space* of $\widetilde{\mathsf{B}}$. Again, we typically have $\mathcal{K} = \{0,1\}^k$ for some positive integer k. Similarly to blockciphers, the invertibility implies that for every $K \in \mathcal{K}$ and $T \in \mathcal{T}$, both $\widetilde{\mathsf{B}}(K,T,\cdot)$ and $\widetilde{\mathsf{B}}^{-1}(K,T,\cdot)$ are necessarily permutations of $\{0,1\}^n$. We use $\widetilde{\mathsf{B}}$ to denote both the tweakable blockcipher and its encryption algorithm.

We measure the security of a tweakable blockcipher through its computational indistinguishability from a random tweakable permutation.

Definition 2.4 ((S)TPRP security)**.** *Given a tweakable blockcipher $\widetilde{B} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$ and an adversary $\mathscr{A}$ that has black-box access to $\widetilde{B}$, we define the advantage of $\mathscr{A}$ in breaking the security of $\widetilde{B}$ in a chosen plaintext attack as*

$$\mathbf{Adv}_{\widetilde{\mathsf{B}}}^{\widetilde{\mathbf{prp}}}(\mathscr{A}) = \Pr\left[K \leftarrow\!\!\$\, \mathcal{K} : \mathscr{A}^{\widetilde{\mathsf{B}}_K(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\widetilde{\pi} \leftarrow\!\!\$\, \widetilde{\mathrm{Perm}}(\mathcal{T}, 2^n) : \mathscr{A}^{\widetilde{\pi}(\cdot,\cdot)} \Rightarrow 1\right].$$

If the advantage $\mathbf{Adv}_{\widetilde{\mathsf{B}}}^{\widetilde{\mathbf{prp}}}(\mathscr{A}) \le \epsilon$ for every adversary $\mathscr{A}$ with running time and query complexity bounded by t and q respectively, we say that $\widetilde{B}$ is a (ϵ, t, q)-secure tweakable pseudorandom permutation (TPRP).

For an adversary $\mathscr{A}$ that has black-box access to $\widetilde{B}$ and $\widetilde{B}^{-1}$, we define the advantage of $\mathscr{A}$ in breaking the security of $\widetilde{B}$ in a chosen ciphertext attack as

$$\mathbf{Adv}_{\widetilde{\mathsf{B}}}^{\widetilde{\mathbf{sprp}}}(\mathscr{A}) = \Pr\left[K \leftarrow\!\!\$\, \mathcal{K} : \mathscr{A}^{\widetilde{\mathsf{B}}_K(\cdot,\cdot),\widetilde{B}_K^{-1}(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\widetilde{\pi} \leftarrow\!\!\$\, \widetilde{\mathrm{Perm}}(\mathcal{T}, 2^n) : \mathscr{A}^{\widetilde{\pi}(\cdot,\cdot),\widetilde{\pi}^{-1}(\cdot,\cdot)} \Rightarrow 1\right].$$

If the advantage $\mathbf{Adv}_{\widetilde{\mathsf{B}}}^{\widetilde{\mathbf{sprp}}}(\mathscr{A}) \le \epsilon$ for every adversary $\mathscr{A}$ with running time bounded by t, and encryption and decryption query complexity bounded by q_e and q_d respectively, we say that $\widetilde{B}$ is a (ϵ, t, q_e, q_d)-secure strong tweakable pseudorandom permutation (STPRP).

Remark 2. *We note that we may simply measure the adversarial resources by $q = q_e + q_d$ for STPRP security.*

Pseudorandom functions. Another frequently used tool in symmetric cryptography is a keyed function that maps inputs from a (potentially infinite) domain $\mathcal{D}$ to a set of fixed-size strings.

A keyed function is an efficient algorithm $F : \mathcal{K} \times \mathcal{D} \to \{0,1\}^n$ that maps a secret key K and an input X to an output string $F(K, X)$. We call $\mathcal{K}$, a finite set, the key space of F and require that n is a positive integer. If $\mathcal{D} = \{0,1\}^m$ with $m > n$, we call F a *compression function.* We let $F_K(X) = F(K, X)$.

We quantify the security of keyed functions through their indistinguishability from a random function (RF) of the same signature. Intuitively, a good keyed function should produce outputs that look "random" and independent of the inputs. We note that the distribution of a "uniformly" chosen function $F : \mathcal{D} \to \{0,1\}^n$ is meaningful even if the domain $\mathcal{D}$ is not finite; for every preimage $X \in \mathcal{D}$, the distribution of the image $F(X)$ is independent and uniform in $\{0,1\}^n$.

Definition 2.5 (PRF security)**.** *Given a keyed function $F : \mathcal{K} \times \mathcal{D} \to \{0,1\}^n$ and an adversary $\mathscr{A}$ that has black-box access to F, we define the advantage of $\mathscr{A}$ in breaking the security of F as*

$$\mathbf{Adv}_F^{\mathbf{prf}}(\mathscr{A}) = \Pr\left[K \leftarrow\$\, \mathcal{K} : \mathscr{A}^{F_K(\cdot)} \Rightarrow 1\right] - \Pr\left[f \leftarrow\$\, \mathrm{Func}(\mathcal{D}, 2^n) : \mathscr{A}^{f(\cdot)} \Rightarrow 1\right].$$

If the advantage $\mathbf{Adv}_F^{\mathbf{prf}}(\mathscr{A}) \leq \epsilon$ for every adversary $\mathscr{A}$ with time complexity, query complexity and data complexity (in bits) of all its queries limited by t, q and σ respectively, we say that F is a (ϵ, t, q, σ)-secure pseudorandom function (PRF)*.*

Remark 3. *We note that the accounting of resources for PRF security of keyed functions may differ slightly from what we stated in Definition 2.5. In particular, we may measure data complexity in blocks of bits rather than bits (e.g. for blockcipher-based construction), or we may add other resource parameters (such as maximal length of any query).*

Tweakable keyed functions While it is possible to define the security of a tweakable PRF, the security notion would completely overlap with the notion of a PRF. This is because the idealized reference object, a "tweakable" random function, is the same as a random function from the set of function whose domain is augmented by a set of tweaks.

However, we sometimes refer to keyed functions (and random functions with the same signature) as "tweakable" to emphasize that their domains are of the form $\mathcal{T} \times \mathcal{D}$ for some sets $\mathcal{T}$ and $\mathcal{D}$ and their inputs *logically* consist of "tweaks" $\mathsf{T} \in \mathcal{T}$ and "data" $X \in \mathcal{D}$. For a tweakable keyed function $F : \mathcal{K} \times (\mathcal{T} \times \mathcal{D}) \to \mathcal{R}$, we let $F_K^T(X) = F_K(T, X) = F(K, T, X)$.

RP-RF switch. One of the best known results in provable security says that a PRP secure blockcipher is also a provably secure PRF with the same signature, albeit with a birthday-bounded loss of security [BR06]. This is formalized in Lemma 2.6, where we only focus on the information-theoretic part of the problem.

Lemma 2.6 (RP-RF switch)**.** *Let n be a positive integer, and let $\mathscr{A}$ be an information theoretic adversary that makes no more than q queries. Then*

$$\Pr\left[\pi \leftarrow\!\!\$ \operatorname{Perm}(2^n) : \mathscr{A}^{\pi(\cdot)} \Rightarrow 1\right] - \Pr\left[f \leftarrow\!\!\$ \operatorname{Func}(2^n) : \mathscr{A}^{f(\cdot)} \Rightarrow 1\right] \leq \frac{q^2}{2^{n+1}}.$$

Almost universal and almost XOR universal hash functions. Both almost universal and almost XOR universal hash functions (AXU) are keyed functions that possesses certain statistical properties.

An ϵ-almost universal hash function is a keyed function $H : \mathcal{K} \times \mathcal{M} \to \{0,1\}^n$ with $\mathcal{M} \subset \{0,1\}^*$ that maps a key K and a string M to a hash $H(K, M)$, such that the distribution of hashes (defined over the randomness of the key) is almost uniform. That is, we have for every $M \in \mathcal{M}$ and every $h \in \{0,1\}^n$ that

$$\Pr\left[K \leftarrow\!\!\$ \mathcal{K} : H(M) = h\right] \leq \epsilon.$$

An ϵ'-AXU hash function is a keyed function $H' : \mathcal{K}' \times \mathcal{M}' \to \{0,1\}^{n'}$ with $\mathcal{M}' \subset \{0,1\}^*$ that maps a key K' and a string M' to a hash $H'(K', M')$, such that the distribution of xors of hashes (defined over the randomness of the key) is almost uniform. That is, we have for every $M'_1 \neq M'_2 \in \mathcal{M}'$ and every $h' \in \{0,1\}^{n'}$ that

$$\Pr\left[K' \leftarrow\!\!\$ \mathcal{K}' : H'(M'_1) \oplus H'(M'_2) = h'\right] \leq \epsilon'.$$

We note that a keyed function can be ϵ-almost universal and ϵ'-AXU at the same time for two (potentially different) smallest values ϵ and ϵ'.

2.4 Authenticated Encryption

In this section, we state the definition of the most common syntax and security notions of AE schemes.

Syntax. We follow the four-input syntax for AE schemes with associated data as defined by Rogaway [Rog02]. A scheme for authenticated encryption (AE) with associated data (AD) is a triple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The key space $\mathcal{K}$ is a set endowed with a probabilistic distribution, and $\mathcal{E}$ and $\mathcal{D}$ are two efficient, deterministic algorithms. If $\mathcal{K}$ is finite and it is not said otherwise, we assume the distribution is uniform. The encryption algorithm $\mathcal{E} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C}$ maps a key $K \in \mathcal{K}$, a nonce[4] $N \in \mathcal{N}$, AD $A \in \mathcal{A}$ and a plaintext (or a message) $M \in \mathcal{M}$ to a ciphertext $C \in \mathcal{C}$. The sets $\mathcal{N}$, $\mathcal{A}$, $\mathcal{M}$ and $\mathcal{C}$ are respectively called nonce space, AD space, message space, and ciphertext space. They are all subsets of $\{0,1\}^*$.[5] The decryption algorithm $\mathcal{D} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C} \to \mathcal{M} \times \{\perp\}$ maps a key $K \in \mathcal{K}$, a nonce $N \in \mathcal{N}$, AD $A \in \mathcal{A}$ and a ciphertext $C \in \mathcal{C}$ to either a

[4]We understand the term "nonce" as a value that ought not to repeat, but may (and if it does, it is considered a misuse of the "nonce").

[5]It may seem unfair that only the key space gets to be included in Π explicitly, while the nonce, AD and message spaces do not. Perhaps it is because $\mathcal{K}$ is the equivalent of the key generator algorithm.

plaintext $M \in \mathcal{M}$, or to the distinguished symbol $\perp$ used to signal an authentication error. We let

$$\mathcal{E}_K^{N,A}(M) = \mathcal{E}_K(N, A, M) = \mathcal{E}(K, N, A, M) \text{ and } \mathcal{D}_K^{N,A}(C) = \mathcal{D}_K(N, A, C) = \mathcal{D}(K, N, A, C).$$

To avoid pathological corner-cases, we require for every $M \in \{0,1\}^*$ that if $M \in \mathcal{M}$ then $M' \in \mathcal{M}$ for all $M' \in \{0,1\}^{|M|}$. We also require that for all $K \in \mathcal{K}$, $N \in \mathcal{N}$, $A \in \mathcal{A}$ and $M \in \mathcal{M}$, the ciphertext expansion (a.k.a. stretch) is positive and only depends on the AD and the length of the plaintext, i.e., $|\mathcal{E}(K, N, A, M)| = |M| + \lambda(A, |M|)$ for some function $\lambda : \mathcal{A} \times \mathbb{N} \to \mathbb{N}$.

We note that in most cases, the ciphertext expansion is a non-negative constant τ, i.e., we have $\lambda(A, m) = \tau$ for any (A, m). We will assume that the stretch is equal to a non-negative constant τ unless explicitly stated otherwise. We further note that the ciphertext C is often composed of a core ciphertext C' that encrypts the message and an authentication tag T, i.e., $\mathcal{E}(K, N, A, M) = C' \| T$ with $|C'| = |M|$ and $|T| = \tau$.

We finally require that Π meets the *correctness* requirement: for every $K \in \mathcal{K}$, $N \in \mathcal{N}$, $A \in \mathcal{A}$ and $M \in \mathcal{M}$, we must have $\mathcal{D}(K, N, A, \mathcal{E}(K, N, A, M)) = M$. In other words, any valid ciphertext must always decrypt correctly.

As we will almost exclusively discuss AE schemes that can process associated data in this thesis, we will refer to AE schemes with associated data simply as AE schemes. In the case that a message-only AE scheme will be considered, this will be made clear explicitly.

Security of nonce-based AE schemes. A secure nonce-based AE scheme provides strong confidentiality and authenticity guarantees, as long as the user of the scheme ensures that each call to the encryption algorithm is done with a unique nonce. There are two security definitions for nonce based AE schemes that formalize this intuition. They can be seen as two variants of the same notion however, as they are shown equivalent (see Lemma 2.9).

In both of these security notions, the requirement about the freshness of the nonces is modelled as an assumption about the adversary, who is thought to be *nonce-respecting*. A nonce-respecting adversary is any algorithm that, given oracle access to a keyed instance of a scheme Π, uses a fresh nonce with each query to the encryption oracle it makes. We stress that the adversary is free to repeat nonces in decryption queries.

Two-requirement nonce-based AE security. The original *two-requirement* variant formalized by Rogaway [Rog02] consists of two standalone definitions. The confidentiality of a scheme Π is captured through indistinguishability of ciphertexts from random strings in a CPA by a nonce-respecting adversary, formalized in games **priv-R** and **priv-I** defined in Figure 2.1.

The authenticity is formalized as integrity (unforgeability) of ciphertexts, where an adversary plays the game **auth** defined in Figure 2.1 and its goal is to find a new ciphertext tuple that decrypts correctly.

Definition 2.7 (PRIV and AUTH AE security [Rog02])**.** *Given a nonce-based AE scheme* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *with a constant ciphertext expansion* τ *and an adversary* $\mathscr{A}$*, we define the advantage of* $\mathscr{A}$ *in breaking the confidentiality of* Π *in a chosen plaintext attack (with help of the games* **priv-*R*** *and* **priv-*I*** *in Figure 2.1) as*

$$\mathbf{Adv}^{\mathbf{priv}}_{\Pi}(\mathscr{A}) = \Pr[\mathscr{A}^{\mathbf{priv}\text{-}\boldsymbol{R}_\Pi} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{priv}\text{-}\boldsymbol{I}_\Pi} \Rightarrow 1].$$

We define the advantage of an adversary $\mathscr{A}'$ *in breaking the authenticity of* Π *in a chosen ciphertext attack (wit help of the game* **auth** *in Figure 2.1) as*

$$\mathbf{Adv}^{\mathbf{auth}}_{\Pi}(\mathscr{A}') = \Pr[\mathscr{A}'^{\mathbf{auth}_\Pi} \text{ forges}]$$

where "$\mathscr{A}'$ forges*" denotes the event that the* Dec *oracle returns a value different from* $\perp$*.*

If $\mathbf{Adv}^{\mathbf{priv}}_{\Pi}(\mathscr{A}) \leq \epsilon$ *for all adversaries* $\mathscr{A}$ *whose running time is limited by* t*, and whose query complexity and data complexity (in bits) in all queries is limited by* q *and* σ *respectively then we say that* Π *is a* (ϵ, t, q, σ)*-PRIV secure nonce-based AE scheme.*

If $\mathbf{Adv}^{\mathbf{auth}}_{\Pi}(\mathscr{A}') \leq \epsilon'$ *for all adversaries* $\mathscr{A}'$ *whose running time is limited by* t'*, and whose encryption and decryption query complexity is bounded by* q'_e *and* q'_d *respectively, and whose data complexity (in bits) in all queries is limited by* σ' *then we say that* Π *is a* $(\epsilon', t', q'_e, q'_d, \sigma')$*-AUTH secure nonce-based AE scheme.*

Remark 4. *We note that the accounting of resources of an adversary against an AE scheme may slightly differ from what we state in the security definitions. For example we may count the total number of queries* $q = q_e + q_d$ *only, or keep track of data complexities in encryption and decryption queries separately. We may also measure the data complexities in blocks of bits rather than bits, or introduce new parameters, such as maximal lengths of queries.*

This applies to all definitions of AE security in this section.

All-in-one nonce-based AE security. The second variant of the nonce-based AE security notion captures both confidentiality and authenticity of the scheme in a single *all-in-one* security definition [RS06b]. The notion is based on the indistinguishability of the real scheme from a pair of oracles that idealize the security properties of AE, a random-strings oracle for encryption queries, and an always-reject oracle for decryption queries. This is formally defined by the games **nae-R** and **nae-I** in Figure 2.2.

Definition 2.8 (NAE security [RS06b])**.** *Given a nonce-based AE scheme* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *with a constant ciphertext expansion* τ *and an adversary* $\mathscr{A}$*, we define the advantage of* $\mathscr{A}$ *in breaking the AE security of* Π *in a chosen ciphertext attack (with help of the games* **nae-*R*** *and* **nae-*I*** *in Figure 2.2) as*

$$\mathbf{Adv}^{\mathbf{nae}}_{\Pi}(\mathscr{A}) = \Pr[\mathscr{A}^{\mathbf{nae}\text{-}\boldsymbol{R}_\Pi} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{nae}\text{-}\boldsymbol{I}_\Pi} \Rightarrow 1].$$

```
proc initialize                         priv-R_Π
$K \leftarrow_{\$} \mathcal{K}$
$\mathcal{X} \leftarrow \varnothing$

proc Enc($N, A, M$)
if $N \in \mathcal{X}$ then
    return $\bot$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{N\}$
$C \leftarrow \mathcal{E}(K, N, A, M)$
return $C$
```

```
proc initialize                         priv-I_Π
$\mathcal{X} \leftarrow \varnothing$

proc Enc($N, A, M$)
if $N \in \mathcal{X}$ then
    return $\bot$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{N\}$
$C \leftarrow_{\$} \{0,1\}^{|M|+\tau}$
return $C$
```

```
proc initialize                         auth_Π
$K \leftarrow_{\$} \mathcal{K}$
$\mathcal{X} \leftarrow \varnothing$, $\mathcal{Y} \leftarrow \varnothing$

proc Enc($N, A, M$)
if $N \in \mathcal{X}$ then
        return $\bot$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{N\}$
$C \leftarrow \mathcal{E}(K, N, A, M)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(N, A, C)\}$
return $C$

proc Dec($N, A, C$)
if $(N, A, C) \in \mathcal{Y}$ then
        return $\bot$
return $\mathcal{D}(K, N, A, C)$
```

Figure 2.1 – **Two-requirement definition of NAE security** for a scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with ciphertext expansion τ.

If $\mathbf{Adv}_{\Pi}^{\mathbf{nae}}(\mathscr{A}) \leq \epsilon$ *for all adversaries* $\mathscr{A}$ *whose running time is limited by* t*, and whose encryption and decryption query complexity is bounded by* q_e *and* q_d *respectively, and whose data complexity (in bits) in all queries is limited by* σ*, then we say that* Π *is an* $(\epsilon, t, q_e, q_d, \sigma)$*-secure nonce-based AE scheme.*

Equivalence of the all-in-one and two-requirement nonce-based AE notions. Rogaway and Shrimpton, who introduced the all-in-one definition of AE security, proved that the PRIV+AUTH security is equivalent with the NAE security. That is, if both the PRIV advantage and the AUTH advantage is small for all reasonable adversaries, then necessarily must also be the NAE advantage, and similar inequalities apply in the opposite direction. This is expressed formally in Lemma 2.9.

Lemma 2.9 (Corollary of Propositions 8 and 9 [RS06b]). *Let* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *be a nonce-based AE scheme, and* $\mathscr{A}$ *be an adversary that runs in time* t *and asks* q_e *and* q_d *encryption and decryption queries, respectively, that have a total data complexity of* σ *bits. Then we have*

$$\mathbf{Adv}_{\Pi}^{\mathbf{nae}}(\mathscr{A}) \leq \mathbf{Adv}_{\Pi}^{\mathbf{priv}}(\mathscr{B}) + \mathbf{Adv}_{\Pi}^{\mathbf{auth}}(\mathscr{C})$$

```
proc initialize                                   $\boxed{\textbf{nae-R}_\Pi}$
$K \leftarrow_\$ \mathcal{K}$
$\mathcal{X} \leftarrow \varnothing$, $\mathcal{Y} \leftarrow \varnothing$

oracle Enc$(N, A, M)$
if $N \in \mathcal{X}$ then
    return $\bot$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{N\}$
$C \leftarrow \mathcal{E}(K, N, A, M)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(N, A, C)\}$
return $C$

oracle Dec$(N, A, C)$
if $(N, A, C) \in \mathcal{Y}$ then
    return $\bot$
return $\mathcal{D}(K, N, A, C)$
```

```
proc initialize                                   $\boxed{\textbf{nae-I}_\Pi}$
$\mathcal{X} \leftarrow \varnothing$

oracle Enc$(N, A, M)$
if $N \in \mathcal{X}$ then
    return $\bot$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{N\}$
$C \leftarrow_\$ \{0,1\}^{|M|+\tau}$
return $C$

oracle Dec$(N, A, C)$
return $\bot$
```

Figure 2.2 – **All-in-one definition of NAE security** for a scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with ciphertext expansion τ.

for some $\mathscr{B}$ that runs in time bounded by $t + \gamma_1 \cdot (q_e + \sigma)$ for a positive constant γ_1 and asks q_e queries of no more than σ bits, and some $\mathscr{C}$ that runs in time bounded by $t+\gamma_2 \cdot (q_e+q_d+\sigma)$ for a positive constant γ_2 and asks q_e and q_d encryption and decryption queries, respectively, of no more than σ bits in total.

Let further $\mathscr{B}'$ be an adversary that runs in time t'_1, asks q encryption queries that have a total data complexity of σ'_e bits, and $\mathscr{C}'$ be an adversary that runs in time t'_2 and asks q'_e and q'_d encryption and decryption queries, respectively, that have a total data complexity of σ' bits. Then we have

$$\mathbf{Adv}_\Pi^{\mathbf{priv}}(\mathscr{B}') \leq \mathbf{Adv}_\Pi^{\mathbf{nae}}(\mathscr{A}'_1) \quad \textit{and} \quad \mathbf{Adv}_\Pi^{\mathbf{auth}}(\mathscr{C}') \leq \mathbf{Adv}_\Pi^{\mathbf{nae}}(\mathscr{A}'_2)$$

for some $\mathscr{A}'_1$ that runs in time bounded by $t'_1 + \gamma'_1 \cdot (q' + \sigma'_e)$ for a positive constant γ'_1 and asks q' encryption queries of no more than σ'_e bits in total, and some $\mathscr{A}'_2$ that runs in time bounded by $t'_2 + \gamma'_2 \cdot (q'_e + q'_d + \sigma')$ for a positive constant γ'_2 and asks q'_e and q'_d encryption and decryption queries, respectively, of no more than σ' bits in total.

Nonce misuse-resistant AE security. In 2006, Rogaway and Shrimpton pointed out that while simple and generally reasonable, the requirement that nonces do not repeat for encryption queries cannot always be met in practice [RS06b]. This may happen e.g. due to an erroneous implementation, a looping counter, insufficient entropy when using a random value as the nonce, or when cloning a virtual machine with a long-term symmetric key. A nonce repetition meant a complete loss of security for all nonce-based AE schemes at the time.

Rogaway and Shrimpton proposed to design AE schemes that achieve the NAE security with unique nonces, and that only suffer the unavoidable loss of security when the nonces repeat; that the ciphertexts repeat whenever the full triple of inputs is repeated. This intuition is formalized in the security notion of *nonce misuse-resistant AE* (MRAE) using the games **mrae-R** and **mrae-I** in Figure 2.3. These two games are almost identical to those used to define NAE security (see Definition 2.8), except that now the adversary can repeat nonces, but is forbidden to repeat a triple (N, A, M), for which it knows the reply trivially.

Definition 2.10 (MRAE security [RS06b])**.** *Given a nonce-based AE scheme* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *with a constant ciphertext expansion* τ *and an adversary* $\mathscr{A}$*, we define the advantage of* $\mathscr{A}$ *in breaking the nonce misuse-resistant AE security of* Π *in a chosen ciphertext attack (with help of the games* **mrae-*R*** *and* **mrae-*I*** *in Figure 2.3) as*

$$\mathbf{Adv}_{\Pi}^{\mathbf{mrae}}(\mathscr{A}) = \Pr[\mathscr{A}^{\mathbf{mrae}\text{-}\boldsymbol{R}_{\Pi}} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{mrae}\text{-}\boldsymbol{I}_{\Pi}} \Rightarrow 1].$$

If $\mathbf{Adv}_{\Pi}^{\mathbf{mrae}}(\mathscr{A}) \leq \epsilon$ *for all adversaries* $\mathscr{A}$ *whose running time is limited by* t*, and whose encryption and decryption query complexity is bounded by* q_e *and* q_d *respectively, and whose data complexity (in bits) in all queries is limited by* σ *then we say that* Π *is a* $(\epsilon, t, q_e, q_d, \sigma)$*-secure MRAE scheme.*

```
proc initialize                         $\boxed{\mathbf{mrae\text{-}R}_\Pi}$
$K \leftarrow\!\$\ \mathcal{K}$
$\mathcal{X} \leftarrow \varnothing$, $\mathcal{Y} \leftarrow \varnothing$

oracle Enc($N, A, M$)
if $(N, A, M) \in \mathcal{X}$ then
  return $\perp$
$C \leftarrow \mathcal{E}(K, N, A, M)$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{(N, A, M)\}$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(N, A, C)\}$
return $C$

oracle Dec($N, A, C$)
if $(N, A, C) \in \mathcal{Y}$ then
  return $\perp$
return $\mathcal{D}(K, N, A, C)$
```

```
proc initialize                         $\boxed{\mathbf{mrae\text{-}I}_\Pi}$
$\mathcal{X} \leftarrow \varnothing$

oracle Enc($N, A, M$)
if $(N, A, M) \in \mathcal{X}$ then
    return $\perp$
$C \leftarrow\!\$\ \{0,1\}^{|M|+\tau}$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{(N, A, M)\}$
return $C$

oracle Dec($N, A, C$)
return $\perp$
```

Figure 2.3 – **Nonce misuse-resistant AE (MRAE) security** game for a scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with ciphertext expansion τ.

Rogaway and Shrimpton pointed out that an MRAE scheme can alternatively be characterized as a *pseudo-random injection* (PRI) with constant stretch that is tweaked by the nonce and the associated data.

Informally, the best possible instance of an MRAE scheme would realize an independent random injective mapping from $\{0,1\}^m$ to $\{0,1\}^{m+\tau}$ for every m s.t. $\{0,1\}^m \subseteq \mathcal{M}$ and for every nonce $N \in \mathcal{N}$ and AD $A \in \mathcal{A}$. The injectivity is required for the correctness of the decryption, and the encryption should otherwise be random. We can then measure the security of an actual MRAE scheme through its indistinguishability from a tweakable random injection with the same signature.

Definition 2.11 (PRI security [RS06b]). *Given a nonce-based AE scheme* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *with a constant ciphertext expansion* τ *and an adversary* $\mathscr{A}$*, we define the advantage of* $\mathscr{A}$ *in breaking the pseudo-random injection security of* Π *in a chosen ciphertext attack (with help of the games* **pri-*R*** *and* **pri-*I*** *in Figure 2.4) as*

$$\mathbf{Adv}^{\mathbf{pri}}_{\Pi}(\mathscr{A}) = \Pr[\mathscr{A}^{\mathbf{pri}\text{-}\boldsymbol{R}_\Pi} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{pri}\text{-}\boldsymbol{I}_\Pi} \Rightarrow 1].$$

If $\mathbf{Adv}^{\mathbf{pri}}_{\Pi}(\mathscr{A}) \leq \epsilon$ *for all adversaries* $\mathscr{A}$ *whose running time is limited by* t*, and whose encryption and decryption query complexity is bounded by* q_e *and* q_d *respectively, and whose data complexity (in bits) in all queries is limited by* σ *then we say that* Π *is a* $(\epsilon, t, q_e, q_d, \sigma)$*-secure PRI.*

```
proc initialize                              pri-R_Π
$K \leftarrow\$ \mathcal{K}$

oracle Enc($N, A, M$)
return $\mathcal{E}(K, N, A, M)$

oracle Dec($N, A, C$)
return $\mathcal{D}(K, N, A, C)$
```

```
proc initialize                              pri-I_Π
for $N, A \in \mathcal{N} \times \mathcal{A}$ do
    $f_{N,A} \leftarrow\$ \mathrm{Inj}(\tau)$

oracle Enc($N, A, M$)
return $f_{N,A}(M)$

oracle Dec($N, A, C$)
if $\exists M \in \mathcal{M}$ s.t. $f_{N,A}(M) = C$ then
    return $M$
else
    return $\perp$
```

Figure 2.4 – **Pseudo-random injection (PRI) security** game for a scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with ciphertext expansion τ.

Rogaway and Shrimpton showed that with growing stretch τ, the notions of MRAE and PRI converge. This is because the MRAE notion is aspirational, in the sense that it is impossible for an AE scheme that meets the correctness requirement to have an MRAE advantage of 0; the ciphertexts cannot be uniform because of the necessary injectivity of encryption, and valid ciphertext tuples must exist (so it is impossible to reject every possible adversarial forgery attempt).

This intuition is formally stated in Lemma 2.12. We note that the quantitative difference between the MRAE and the PRI advantage vanishes with increasing τ. The larger

τ, the less likely we are observe collisions in random strings with length $\geq \tau$ and the less likely it is to find an image of a random injection.

Lemma 2.12 (Theorem 7 [RS06b])**.** *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a nonce-based AE scheme, with stretch τ and $\mathscr{A}$ be an adversary that asks q_e and q_d encryption and decryption queries, respectively, so $q = q_e + q_d$ queries in total. Let further $s = \min_{M \in \mathcal{M}}(|M|)$ be the length of the shortest possible plaintext. Then we have*

$$\left|\mathbf{Adv}_{\Pi}^{\mathbf{mrae}}(\mathscr{A}) - \mathbf{Adv}_{\Pi}^{\mathbf{pri}}(\mathscr{A})\right| \leq \frac{q^2}{2^{2+\tau+1}} + \frac{4 \cdot q_d}{2^{\tau}}.$$

Part I

Constructions

Chapter 3

OMD: a Compression Function-based Scheme for Authenticated Encryption

This chapter is dedicated to Offset Merkle-Damgård (OMD), which is the first dedicated scheme for authenticated encryption based on a compression function.

The work presented in this chapter is a result of joint work with Simon Cogliani, Diana-Stefania Maimut, David Naccache, Rodrigo Portella do Canto, Reza Reyhanitabar and Serge Vaudenay which was published in SAC 2014 [CMN+14].

The implementations presented in Section 3.7 were developed by Robin Ankele and Ralph Ankele [AA14], Johan Droz [Dro15], and Martin Georgiev [Geo15] as parts of student projects (co-)supervised by the author of this thesis.

Organization of the Chapter. We start with a brief overview of the related work in Section 3.1 and a summary of the contribution in Section 3.2.

We introduce OMD in Section 3.3. We give a description of OMD in Section 3.4 and discuss OMD as a candidate in the CAESAR competition in Section 3.5. We give the security analysis of OMD in Section 3.6 and discuss its performance in Section 3.7.

3.1 Related Work

OMD follows the four-input syntax for nonce-based AE schemes with associated data by Rogaway [Rog02] (see Section 2.4).

The security notion targeted by OMD is the nonce-based AE security proposed by Rogaway [Rog02] (see Section 2.4). The core of the encryption in OMD is inspired by the Merkle-Damgård hash construction [Dam89, Mer89], while the use of whitening offsets is mainly inspired by OCB [RBBK01, Rog04a, KR11]. The recommended instances of OMD are based on the compression functions of the standard hash functions SHA256 and SHA512 [oST12]. The assumption that when keyed, the compression functions of the SHA2-family are secure PRFs is not unprecedented; for example Bellare uses a similar assumption on SHA1 to prove the security of NMAC and HMAC [Bel06a].

3.2 Contribution

In this chapter, we present a novel, dedicated AE scheme called Offset Merkle-Damgård, a mode of operation for a keyed compression function. OMD competed in the CAESAR competition and finished as a second round candidate.

We show that OMD is NAE secure, assuming the underlying compression function is a secure PRF. We also investigate the performance of optimized software implementations of OMD.

To our best knowledge, OMD is the first AE scheme based on a (keyed) compression function; this work adds compression functions to the list of low-level primitives that can be used to construct practical and secure AE schemes. We thus contribute to cryptographic diversity, which helps to limit the global impact of devastating attacks.

Beyond demonstrating the mere feasibility of AE based on compression functions, the instances of OMD based on the compression functions of SHA256 and SHA512 also provide a large quantitative security margin while delivering a decent performance, especially on high-end CPU's with suitable instruction extension sets.

3.3 Offset Merkle-Damgård

Offset Merkle-Damgård (OMD) is a nonce-based AE scheme. Unlike the majority of other AE schemes (that were designed before and during the CAESAR competition), which are either blockcipher-based or permutation-based, OMD is designed as a mode of operation for a *compression function.* The motivation for using a compression function as a low-level primitive is manifold:

1. the cryptographic community has spent more than two decades on public research and standardization activities on hash functions resulting to development of a rich source of secure and efficient compression functions;

2. the standard SHA family of algorithms is heavily employed in many of the most common cryptographic applications and one can easily use off-the-shelf, highly optimized implementations of these functions [GYG12, GCG12];

3. Intel has recently introduced new instruction extensions that support performance acceleration of SHA-1 and SHA-256 on next-generation processors [GGY$^+$13];

4. having a set of AE schemes based on a large number of different primitives contributes to *cryptographic diversity*, which we feel is of importance, as an almost exclusive use of a single primitive can have catastrophic consequences if the security of that primitive collapses;

5. we believe that having a diverse set of AE schemes, based on different primitives, can be interesting from a practical viewpoint, providing the opportunity to choose among the AE algorithms based on what primitives have already been available and implemented on the platform of choice and to reuse such implementations.

We designed OMD to be provably secure, keeping several functional features, as well as performance in mind. Some of the interesting features of OMD, and its instantiations OMD-sha256 and OMD-sha512, are as follows:

Provable Security in the Standard Model. OMD achieves its security goals (confidentiality and authenticity) provably, based on the assumption that its underlying keyed compression function is a PRF, an assumption which is among the well-known and widely-used ones [Bel06b]. From a theoretical point of view this is an advantage compared to permutation-based AE schemes whose security proofs are done in the *ideal permutation* model, and thus have no formal ties with the security of the cryptographic permutation that they actually use.

High Quantitative Security Level. When implemented with an off-the-shelf compression function such as those of the standard SHA family [oST12], OMD can achieve much higher security level compared to AES-based schemes. The proven security of OMD-sha256 and OMD-sha512 falls off in about $\frac{\sigma^2}{2^{256}}$ and $\frac{\sigma^2}{2^{512}}$, respectively, where σ is the total number of calls to the compression function. In comparison, for the same key size and tag size, the proven security of all the standardized blockcipher-based AE schemes using AES (e.g. all five dedicated schemes specified in ISO/IEC 19772:2009) falls off in about $\frac{\sigma'^2}{2^{128}}$ where σ' is the total number of calls to AES. We note that it is possible to get blockcipher-based AE schemes with (high) beyond birthday-bound security, but the existing schemes with beyond birthday-bound security have a degraded efficiency [Iwa06, LPRM07, Iwa08, LST12].

Online. OMD encryption is online; that is, it outputs a stream of ciphertext as a stream of plaintext arrives with a constant latency and using constant memory. After receiving an indication that the plaintext is over, the final part of ciphertext together with the tag is output. OMD decryption is *internally* online: one can generate a stream of plaintext bits as the stream of ciphertext bits comes in, but no part of the plaintext stream will be returned before the whole ciphertext stream is decrypted and the tag is verified to be correct.

Flexible Parameters. OMD-sha256 can support any key length up to 256 bits, tag length up to 256 bits, and nonce length up to 255 bits. OMD-sha512 can support any key length up to 512 bits, tag length up to 512 bits, and nonce length up to 511 bits. These upper bounds on the parameters' length will satisfy the required security level of almost any imaginable application today and well beyond. The lower bounds on the parameters' lengths should be selected based on the specific security level sought by an application; for instance, most applications would not use keys shorter than 128 bits, tags shorter than 32 bits and nonces shorter than 64 bits.

3.4 Description

OMD is has two parameters; to instantiate it, one must specify a keyed compression function $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ and *fix* a tag length $\tau \leq n$. For simplicity, we assume that the key space is $\mathcal{K} = \{0,1\}^k$. We require that $m \leq n$. We let OMD$[F, \tau]$ denote the OMD mode of operation using the keyed compression function F_K and the fixed tag length τ. We note that assuming the input of F to be composed of pairs of n-bit and m-bit blocks (most often chaining and data blocks) is without loss of generality, as a keyed compression function with a monolithic message domain $\{0,1\}^{m+n}$ can be used as one with a structured domain without loss of security.

At the first glance, imposing $m \leq n$ may look odd as compression functions usually have a larger data block length than its output (chaining) block length, but we note that in practice, the compression functions of standard hash functions (e.g. SHA-1 or the SHA-2 family) are keyless. Therefore one needs to use k bits of their b-bit data block to get a keyed function. So, there will be no waste in each call to the compression function if $m = n$ and $b = n + k$; for example, when the key length is 256 bits and the compression function of SHA-256 is used.

An overview. An instance OMD$[F, \tau]$ has a key space $\mathcal{K} = \{0,1\}^k$, and can work with any AD space $\mathcal{A} \subseteq \{0,1\}^{\leq (m+n) \cdot 2^{n-4}}$ and any message space $\mathcal{M} \subseteq \{0,1\}^{\leq m \cdot 2^{n-4}}$. We let $\ell_{max} = \max_{X \in \mathcal{A} \cup \mathcal{M}}(|X|_m)$ denote the upper bound on the maximum number of blocks in any input to the encryption algorithm. The nonce space is the set of ν-bit strings $\mathcal{N} = \{0,1\}^\nu$ for an integer $1 \leq \nu < n$.[1]

The encryption algorithm of OMD$[F, \tau]$ inputs four arguments (a secret key K, a nonce N, associated data A, a message M), and outputs a tagged ciphertext $\mathbb{C} = C \| T \in \{0,1\}^{|M|+\tau}$. The decryption algorithm of OMD$[F, \tau]$ takes as input four arguments (a secret key K, a nonce N, associated data A, a tagged ciphertext $C \| T$), and either outputs the whole message $M \in \{0,1\}^{|C|-\tau}$ at once, or an error message ($\perp$) in case of an authentication error.

Internally, the encryption algorithm splits the message M in blocks $M_1, \ldots, M_\ell \overset{m}{\leftarrow} M$ and processes them using a sequence of chained calls to F similar to the Merkle-Damgård construction for hashing [Dam89, Mer89]. The main differences with this hash construction are that:

- the intermediate chaining values are used as keystream bits for encryption,
- (a part of) the input is masked with key-dependent whitening offsets upon every call to F (hence the name).

[1]In theory, OMD$[F, \tau]$ can be used with a nonce space $\mathcal{N} = \{0,1\}^{<n}$, but we prefer $\mathcal{N} = \{0,1\}^\nu$ for simplicity.

The AD processing is inspired by a Wegman-Carter MAC [CW77, WC81] that uses F to compute an almost XOR universal hash of the AD, and masks the hash with the partial tag computed in the message processing.

Figure 3.1 illustrates the encryption algorithm of OMD$[F, \tau]$. The decryption algorithm is straightforward to derive. It is almost the same as the encryption algorithm, except for a tag comparison (verification) at the end of the decryption process. A formal algorithmic description of OMD$[F, \tau]$ is provided in Figure 3.2.

In the rest of this chapter, when partitioning a string C into blocks of m bits, such that $|C| \not\equiv m \pmod{m}$, we denote the final (incomplete) block interchangeably with C_ℓ and C_*; i.e. $C = C_1 \| \cdots \| C_\ell$ or $C = C_1 \| \cdots \| C_{\ell-1} \| C_*$.

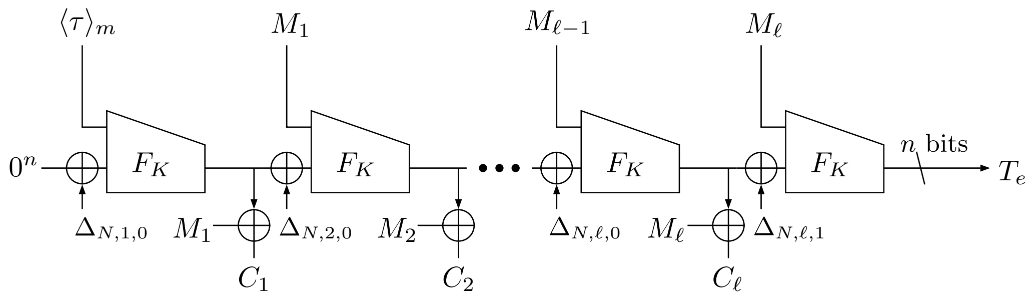
Computing the masking values. As seen from the description of OMD in Figure 3.1, before each call to the underlying keyed compression function we xor a masking value denoted as $\Delta_{N,i,j}$ (the top and middle parts of Figure 3.1) or $\bar{\Delta}_{i,j}$ (the bottom part of Figure 3.1) to the input of F. The purpose of these masks is to randomize each call to F; in Section 3.6 it will be shown that they in fact allow to extend the domain of F. In the following, we describe how these masks are generated.

There are different ways to compute the masking values to satisfy both the security and efficiency criteria [Rog04a, CS08, KR11]. We use the method used in OCB3 [KR11]. In the following, all multiplications are in $\mathsf{GF}(2^n)$.

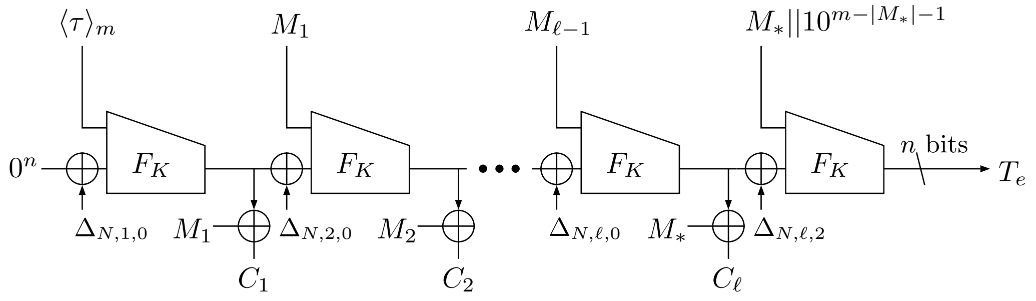
Initialization. In a one-time initialization phase, we compute the derived key L_*, and an array of L_*-dependent values $L[0], \ldots, L[\lceil \log_2(\ell_{max}) \rceil]$. The initial key-dependent value is defined as $L_* = F_K(0^n, 0^m)$. We then let $L[0] = 4 \cdot L_*$, and $L[i] = 2 \cdot L[i-1]$ for $i \geq 1$. We note that the values $L[i]$ can be preprocessed and stored (for a fast implementation) in a table of $\lceil \log_2(\ell_{max}) \rceil$ entries, where ℓ_{max} is the bound on the maximum number of blocks in any message or AD. Alternatively, (if there is a memory restriction) they can be computed on-the-fly for $i \geq 1$. It is also possible to precompute and store some values and then compute the others as needed on-the-fly.

Masking sequence for processing the message. We compute the masks $\Delta_{N,i,j}$ for every query. We define the initial mask $\Delta_{N,0,0} = F_K(N \| 10^{n-1-\nu}, 0^m)$. Then for $i \geq 1$ we let
$\Delta_{N,i,0} = \Delta_{N,i-1,0} \oplus L[\mathtt{ntz}(i)]$, and
$\Delta_{N,i,1} = \Delta_{N,i,0} \oplus 2 \cdot L_*$, and
$\Delta_{N,i,2} = \Delta_{N,i,0} \oplus 3 \cdot L_*$.

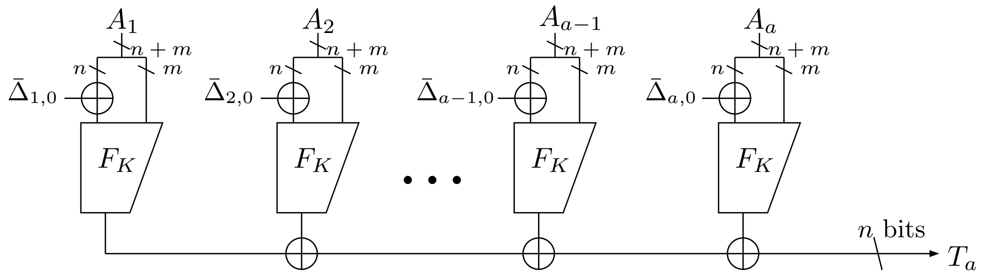
Masking sequence for processing the associated data. We likewise compute the masks $\bar{\Delta}_{i,j}$ for every query. We define $\bar{\Delta}_{0,0} = 0^n$ and let for $i \geq 0$:
$\bar{\Delta}_{i,0} = \bar{\Delta}_{i-1,0} \oplus L[\mathtt{ntz}(i)]$, and
$\bar{\Delta}_{i,1} = \bar{\Delta}_{i,0} \oplus L_*$.



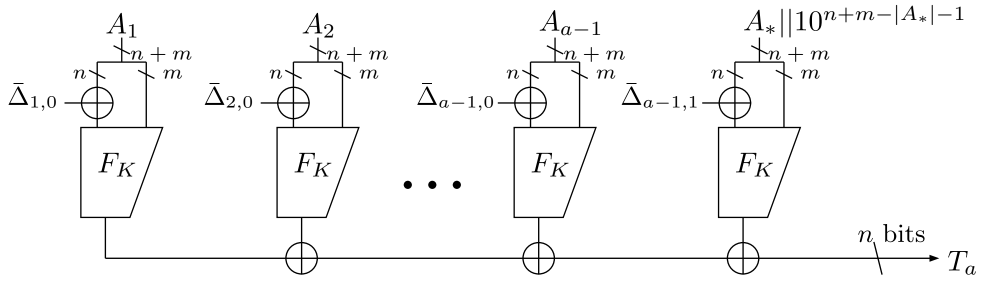
Encrypting a message whose length is a multiple of the block length. No padding is needed.



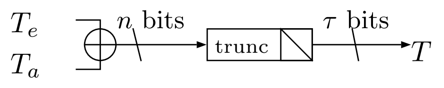
Encrypting a message whose length is not a multiple of the block length. The final message block is padded to make it a full block



Computing T_a for an associated data whose length is a multiple of the input length (i.e $|A_a| = n + m$).



Computing T_a for an associated data whose length is not a multiple of the input length. The final block is padded to make it a full block .



The T is computed as XOR of T_e and T_a truncated to τ bits.

Figure 3.1 – **The encryption algorithm of OMD**$[F, \tau]$ using a keyed compression function $F_K : (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ and a fixed tag length τ, where $m \leq n$. We refer the reader to Section 2.1 regarding the convention for the xor operation.

```
algorithm Initialize($K$)
    $L_* \leftarrow F_K(0^n, 0^m)$
    $L[0] \leftarrow 4 \cdot L_*$
    for $i \leftarrow 1$ to $\lceil \log_2(\ell_{max}) \rceil$ do
        $L[i] = 2 \cdot L[i-1]$
    end for
    return
end algorithm

algorithm HASH$_K(A)$
    $b \leftarrow n + m$
    $A_1 \| A_2 \cdots A_{\ell-1} \| A_\ell \xleftarrow{b} A$
    $T_a \leftarrow 0^n$
    $\Delta \leftarrow 0^n$
    for $i \leftarrow 1$ to $\ell - 1$ do
        $\Delta \leftarrow \Delta \oplus L[\mathtt{ntz}(i)]$
        $\mathsf{Left} \leftarrow A_i[b-1 \cdots m]$
        $\mathsf{Right} \leftarrow A_i[m-1 \cdots 0]$
        $\Phi \leftarrow F_K(\mathsf{Left} \oplus \Delta, \mathsf{Right})$
        $T_a \leftarrow T_a \oplus \Phi$
    end for
    if $|A_\ell| = b$ then
        $\Delta \leftarrow \Delta \oplus L[\mathtt{ntz}(\ell)]$
        $\mathsf{Left} \leftarrow A_\ell[b-1 \cdots m]$
        $\mathsf{Right} \leftarrow A_\ell[m-1 \cdots 0]$
        $\Phi \leftarrow F_K(\mathsf{Left} \oplus \Delta, \mathsf{Right})$
        $T_a \leftarrow T_a \oplus \Phi$
    else
        $\Delta \leftarrow \Delta \oplus L_*$
        $A_{\text{pad}} \leftarrow A_\ell \| 10^{b-|A_\ell|-1}$
        $\mathsf{Left} \leftarrow A_{\text{pad}}[b-1 \cdots m]$
        $\mathsf{Right} \leftarrow A_{\text{pad}}[m-1 \cdots 0]$
        $\Phi \leftarrow F_K(\mathsf{Left} \oplus \Delta, \mathsf{Right})$
        $T_a \leftarrow T_a \oplus \Phi$
    end if
    return $T_a$
end algorithm

algorithm $\mathcal{E}_K(N, A, M)$
    if $\nu > n - 1$ then
        return $\perp$
    end if
    $M_1 \| M_2 \cdots M_{\ell-1} \| M_\ell \xleftarrow{m} M$
    $\Delta \leftarrow F_K(N \| 10^{n-1-\nu}, 0^m)$
    $H \leftarrow 0^n$
    $\Delta \leftarrow \Delta \oplus L[0]$
    $H \leftarrow F_K(H \oplus \Delta, \langle \tau \rangle_m)$
    for $i \leftarrow 1$ to $\ell - 1$ do
        $C_i \leftarrow H \oplus M_i$
        $\Delta \leftarrow \Delta \oplus L[\mathtt{ntz}(i+1)]$
        $H \leftarrow F_K(H \oplus \Delta, M_i)$
    end for
    $C_\ell \leftarrow H \oplus M_\ell$
    if $|M_\ell| = m$ then
        $\Delta \leftarrow \Delta \oplus 2 \cdot L_*$
        $T_e \leftarrow F_K(H \oplus \Delta, M_\ell)$
    else
        $\Delta \leftarrow \Delta \oplus 3 \cdot L_*$
        $M_{\text{pad}} \leftarrow M_\ell \| 10^{m-|M_\ell|-1}$
        $T_e \leftarrow F_K(H \oplus \Delta, M_{\text{pad}})$
    end if
    $T_a \leftarrow \text{HASH}_K(A)$
    $T \leftarrow (T_e \oplus T_a)[n-1 \cdots n-\tau]$
    $\mathbb{C} \leftarrow C_1 \| C_2 \| \cdots \| C_\ell \| T$
    return $\mathbb{C}$
end algorithm

algorithm $\mathcal{D}_K(N, A, \mathbb{C})$
    if $\nu > n - 1$ or $|\mathbb{C}| < \tau$ then
        return $\perp$
    end if
    $C_1 \| C_2 \cdots C_{\ell-1} \| C_\ell \| T \xleftarrow{m} \mathbb{C}$
    $\Delta \leftarrow F_K(N \| 10^{n-1-\nu}, 0^m)$
    $H \leftarrow 0^n$
    $\Delta \leftarrow \Delta \oplus L[0]$
    $H \leftarrow F_K(H \oplus \Delta, \langle \tau \rangle_m)$
    for $i \leftarrow 1$ to $\ell - 1$ do
        $M_i \leftarrow H \oplus C_i$
        $\Delta \leftarrow \Delta \oplus L[\mathtt{ntz}(i+1)]$
        $H \leftarrow F_K(H \oplus \Delta, M_i)$
    end for
    $M_\ell \leftarrow H \oplus C_\ell$
    if $|C_\ell| = m$ then
        $\Delta \leftarrow \Delta \oplus 2 \cdot L_*$
        $T_e \leftarrow F_K(H \oplus \Delta, M_\ell)$
    else
        $\Delta \leftarrow \Delta \oplus 3 \cdot L_*$
        $M_{\text{pad}} \leftarrow M_\ell \| 10^{m-|M_\ell|-1}$
        $T_e \leftarrow F_K(H \oplus \Delta, M_{\text{pad}})$
    end if
    $T_a \leftarrow \text{HASH}_K(A)$
    $T' \leftarrow (T_e \oplus T_a)[n-1 \cdots n-\tau]$
    if $T' = T$ then
        return $M \leftarrow M_1 \| M_2 \| \cdots \| M_\ell$
    else
        return $\perp$
    end if
end algorithm
```

Figure 3.2 – **Definition of OMD[$\boldsymbol{F}, \boldsymbol{\tau}$]**, using a keyed compression function $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \rightarrow \{0,1\}^n$ with $\mathcal{K} = \{0,1\}^k$ and $m \leq n$, and tag length $\tau \leq n$.

3.5 OMD in CAESAR Competition

OMD competed as one of the 57 candidates in the CAESAR competition. It entered the second round, among the 29 second round candidates, and did not advance to the third round. As for all other candidates, a set of concrete instances had to be proposed for OMD. Moreover, both reference, hardware and software implementations needed to be provided for each of these instances.

Recommended instances of OMD. We recommend to instantiate OMD with the compression functions of the SHA-256 and SHA-512 hash functions from NIST FIPS PUB 180-4 [oST12].

Our primary recommendation to instantiate OMD is called OMD-sha256, and uses the underlying compression function of SHA-256 [oST12]. This is intended to be the appropriate choice for implementations on 32-bit machines. The compression function of SHA-256 is a map sha-256 : $\{0,1\}^{256} \times \{0,1\}^{512} \rightarrow \{0,1\}^{256}$. It takes a 256-bit chaining block X and a 512-bit message block Y as input, and it outputs a 256-bit digest Z, i.e. let $Z = \text{sha-256}(X, Y)$.

To use OMD with sha-256, we use the first 256-bit argument X for chaining values as usual. In our notation (see Figure 3.1) this means that $n = 256$. We use the 512-bit argument Y (the message block in sha-256) to input both a 256-bit message block and the key K which can be of any length $k \leq 256$ bits. If $k < 256$ then let the key be $K||0^{256-k}$. That is, we define the keyed compression function $F_K : \{0,1\}^{256} \times \{0,1\}^{256} \rightarrow \{0,1\}^{256}$ needed in OMD as $F_K(H, M) = \text{sha-256}(H, K||0^{256-k}||M)$.

The parameters of OMD-sha256 are as follows:

- The message block length in bits is $m = 256$.
- The key length in bits can be $80 \leq k \leq 256$; but $k < 128$ is not recommended. *If needed*, we pad the key K with 0^{256-k} to make its length exactly 256 bits.
- The nonce (public message number) length in bits can be $96 \leq \nu \leq 255$. We *always* pad the nonce with $10^{255-\nu}$ to make its length exactly 256 bits.
- The associated data block length in bits is $2n = 512$.
- The tag length in bits can be $32 \leq \tau \leq 256$; but it must be noted that the selection of the tag length directly affects the achievable security level (see Section 3.6).

Our secondary recommendation to instantiate OMD is called OMD-sha512, and uses the underlying compression function of SHA-512 [oST12]. This is intended to be the appropriate choice for implementations on 64-bit machines. The compression function of SHA-512 is a map sha-512 : $\{0,1\}^{512} \times \{0,1\}^{1024} \rightarrow \{0,1\}^{512}$. On input a 512-bit chaining block X and a 1024-bit message block Y, it outputs a 512-bit digest $Z = \text{sha-512}(X, Y)$.

To use OMD with sha-512, we use the first 512-bit argument X for chaining values as usual. In our notation (see Figure 3.1) this means that $n = 512$. We use the 1024-bit argument Y (the message block in sha-512) to input both a 512-bit message block and the key K which can be of any length $k \leq 512$ bits. If $k < 512$ then let the key be $K||0^{512-k}$. That is, we define the keyed compression function $F_K : \{0,1\}^{512} \times \{0,1\}^{512} \rightarrow \{0,1\}^{512}$ needed in OMD as $F_K(H, M) = \text{sha-512}(H, K||0^{512-k}||M)$.

The parameters of OMD-sha512 are set as follows:

- The message block length in bits is $m = 512$.
- The key length in bits can be $80 \leq k \leq 512$; but $k < 128$ is not recommended. *If needed*, we pad the key K with 0^{512-k} to make its length exactly 512 bits.
- The nonce (public message number) length in bits can be $96 \leq \nu \leq 511$. We *always* pad the nonce with $10^{511-\nu}$ to make its length exactly 512 bits.
- The associated data block length in bits is $2n = 1024$.
- The tag length in bits can be $32 \leq \tau \leq 512$; but it must be noted that the selection of the tag length directly affects the achievable security level (see Section 3.6).

For both compression functions, we propose named instances that further fix all the remaining parameters. These are listed in Table 3.1. The motivation for the choice of these combinations of parameters is the following. For the key length, we require at least 128 bits, scaling up to the maximal length allowed by each compression function. The nonce length starts at 96 bits, which means 2^{96} distinct nonces. This should suffice for any imaginable application. However, longer nonces are considered too, which can be convenient in some situations, e.g. if the nonce consists of a device-specific prefix and a counter. Choosing the tag length means making a trade-off between security level and efficiency. We therefore provide a relatively broad scale of tag lengths, allowing to tune this trade-off to fit the application. We do not, however, recommend tags shorter than 64 bits.

Instance	Comp. func.	Key len.	Nonce len.	Tag len.
omdsha256k128n96tau128	sha-256	128	96	128
omdsha256k128n96tau64	sha-256	128	96	64
omdsha256k128n96tau96	sha-256	128	96	96
omdsha256k192n104tau128	sha-256	192	104	128
omdsha256k256n104tau160	sha-256	256	104	160
omdsha256k256n248tau256	sha-256	256	248	256
omdsha512k128n128tau128	sha-512	128	128	128
omdsha512k256n256tau256	sha-512	256	256	256
omdsha512k512n256tau256	sha-512	512	256	256

Table 3.1 – **Named instances of OMD** and the values of paramters associated to each instance. The instances are ordered from the primary recommendation to the least recommended instance.

Software and hardware implementations. We provide a reference implementation in the C language for both OMD-sha256 and OMD-sha512. Two optimized implementation for the x86_64 architecture were provided both OMD-sha256 and OMD-sha512. Optimized implementations for the ARM and MIPS architectures were additionally provided for OMD-sha256. The software implementations are further discussed in Section 3.7.

Hardware implementations of OMD-sha256 for the Virtex 6, Virtex 5, Stratix IV and Stratix V FPGA families [Die16] were designed by William Diehl [DG17].

3.6 Security Analysis

In this section, we prove upper bounds on the PRIV and AUTH adversarial advantage for OMD, reducing its security to the security of the underlying compression function as a PRF. We first show that OMD instantiated with a random function provides near-optimal security (because of the ℓ_{max} in the authenticity bound), and then use a sequence of standard hybrid arguments to relate this result to the security of an actual instance. The formal statement about the security of OMD can be found in Theorem 3.1.

Theorem 3.1. *Fix $n \geq 1$ and $\tau \in \{0, 1, \cdots, n\}$. Let $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be a keyed function, where $1 \leq m \leq n$. Let $\mathscr{A}$ be a CPA adversary that runs in time t, makes q_e encryption queries that induce no more than σ_e n calls to F in total, such that no individual input (AD or message) is more that ℓ_{max} m-bit blocks long. Let further $\mathscr{A}'$ be a CCA adversary that runs in time t', makes q'_e encryption queries and q'_d decryption queries that induce no more than σ' calls to F in total, such that no individual input is more that ℓ'_{max} m-bit blocks long. Then*

$$\begin{aligned}
\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{OMD}[F,\tau]}(\mathscr{A}) &\leq \mathbf{Adv}^{\mathbf{prf}}_{F}(\mathscr{B}) + \frac{3\sigma_e^2}{2^n} \\
\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{OMD}[F,\tau]}(\mathscr{A}') &\leq \mathbf{Adv}^{\mathbf{prf}}_{F}(\mathscr{B}') + \frac{3\sigma^2}{2^n} + \frac{q_d \ell_{max}}{2^n} + \frac{q_d}{2^\tau}
\end{aligned}$$

for some $\mathscr{B}$ that makes $2 \cdot \sigma_e$ queries and runs in time $t + \gamma \cdot n \cdot \sigma$ for some constant γ, and $\mathscr{B}'$ that makes $2 \cdot \sigma'$ queries and runs in time $t' + \gamma' \cdot n \cdot \sigma'$ for some constant γ'.

$$\begin{tikzcd}
\mathrm{OMD}[F,\tau] \arrow[r, "\textbf{Lemma 3.4}"] & \mathrm{OMD}[\widetilde{F},\tau] \arrow[r, "\textbf{Lemma 3.3}"] & \overset{\textbf{Lemma 3.2}}{\mathrm{OMD}[\widetilde{R},\tau]}
\end{tikzcd}$$

Figure 3.3 – **An overview of the proof of Theorem 3.1.**

Proof. The proof is obtained by the combing Lemmas 3.2, 3.3 and 3.4, as illustrated in Figure 3.3. $\square$

Generalized OMD using a Random Function. Figure 3.4 shows the $\mathbb{OMD}[\widetilde{R},\tau]$ scheme, which is a generalization of $\text{OMD}[F,\tau]$ that replaces F and the masking offsets by a tweakable random function $\widetilde{R} \leftarrow\$ \widetilde{\text{Func}}(\mathcal{T}, 2^{m+n}, 2^n)$. The tweak space $\mathcal{T}$ consists of five mutually exclusive sets of tweaks; namely, $\mathcal{T} = \mathcal{N} \times \mathbb{N} \times \{0\} \cup \mathcal{N} \times \mathbb{N} \times \{1\} \cup \mathcal{N} \times \mathbb{N} \times \{2\} \cup \mathbb{N} \times \{0\} \cup \mathbb{N} \times \{1\}$. These sets correspond to the sets of labels we used to define the offsets in $\text{OMD}[F,\tau]$. These labels now take on the role of tweaks. More precisely, a call to F masked with $\Delta_{N,i,j}$ is replaced by a call to $\widetilde{R}^{N,i,j}$ and a call to F masked with $\bar{\Delta}_{i,j}$ is replaced by a call to $\widetilde{R}^{i,j}$.

Lemma 3.2. *Let $\mathbb{OMD}[\widetilde{R},\tau]$ be the scheme shown in Figure 3.4. Let $\mathscr{A}$ be an information theoretic CPA adversary that makes q_e encryption queries such that they induce no more than σ_e n calls to F in total, and such that no individual input (AD or message) is more that ℓ_{max} blocks long. Let further $\mathscr{A}'$ be an information theoretic CCA adversary that makes q'_e encryption queries and q'_d decryption queries such that they induce no more than σ' calls to F in total, and such that no individual input is more that ℓ'_{max} blocks long. Then*

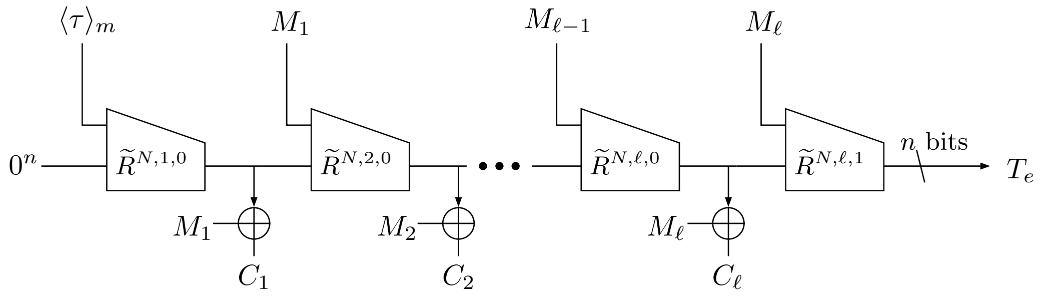
$$\begin{aligned}\mathbf{Adv}^{\mathbf{priv}}_{\mathbb{OMD}[\widetilde{R},\tau]}(\mathscr{A}) &= 0,\\ \mathbf{Adv}^{\mathbf{auth}}_{\mathbb{OMD}[\widetilde{R},\tau]}(\mathscr{A}') &\leq \frac{q_d \ell_{max}}{2^n} + \frac{q_d}{2^\tau}.\end{aligned}$$

Proof. The proof of the privacy bound is straightforward. The adversary $\mathscr{A}$ asks (encryption) queries $(N^1, A^1, M^1) \cdots (N^{q_e}, A^{q_e}, M^{q_e})$ where all N^x values (for $1 \leq x \leq q_e$) are distinct, as the adversary $\mathscr{A}$ is nonce respecting. Referring to Figure 3.4, this means that we are evaluating the function $\widetilde{R}^{N_x,i,j}$ on a single input for each (N^x, i, j), hence the images that the adversary sees (i.e. $\mathbb{C}^x$ for $1 \leq x \leq q_e$) are independent uniformly random values.

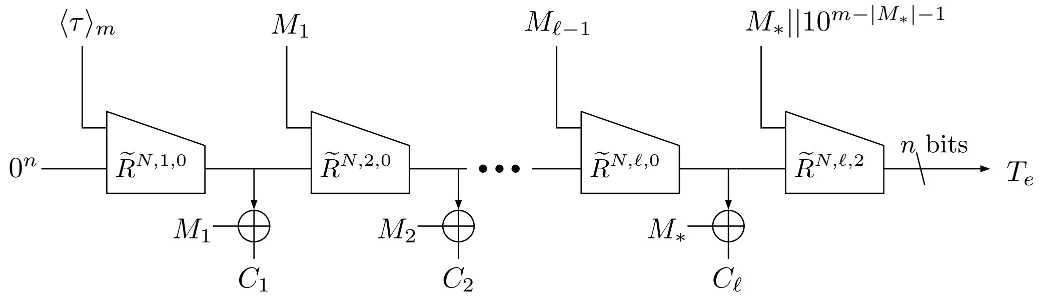
The authenticity bound can be shown by a straightforward, but lengthy case analysis. First we consider the single-decryption-query case where an adversary $\mathscr{A}''$ only makes one decryption (verification) query, and then we use the generic result of Bellare et al. [BGM04] to get a bound against any adversary $\mathscr{A}'$ that makes multiple (say q_d) verification queries.

The adversary $\mathscr{A}''$ makes encryption queries $(N^1, A^1, M^1) \cdots (N^{q_e}, A^{q_e}, M^{q_e})$. We let $M^i = M^i_1 \cdots M^i_{\ell_i}$ or $M^i = M^i_1 \cdots M^i_{\ell_i - 1} M^i_*$ denote the message blocks and $A^i = A^i_1 \cdots A^i_{a_i}$ or $A^i = A^i_1 \cdots A^i_{a_i-1} A^i_*$ be the associated data blocks in the i^{th} query. Let $\mathbb{C}^i = C^i \| T^i$ be the ciphertext returned to $\mathscr{A}''$ upon query (N^i, A^i, M^i). That is, we use superscripts to indicate query numbers and subscripts to denote the block indices in each query. We additionally let H^x_i denote the chaining value that is fed to $\widetilde{R}^{N^x,i+1,0}$ along with M^x_i for $i = 1, \ldots, \ell - 1$. The value H_ℓ is fed to $R^{N^x,\ell,j}$ for $j \in \{1,2\}$ and $H_0 = 0^n$. A special case occurs if $|M^x| = 0$; then H_1 is fed to $R^{N^x,0,2}$ along with the string 10^{m-1}.

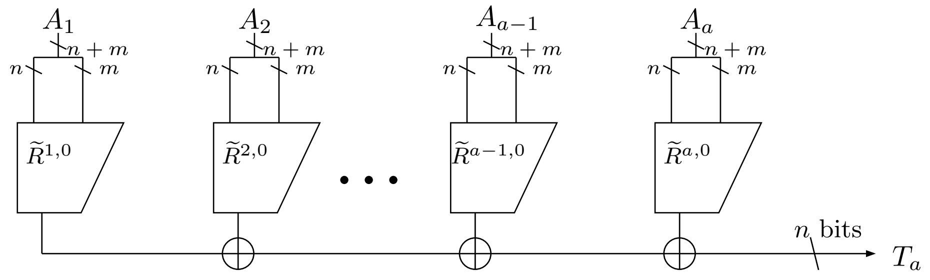
Let $(N, A, \mathbb{C})$ be the forgery attempt by $\mathscr{A}''$, where $N \in \{0,1\}^\nu$ is the nonce, $A =$



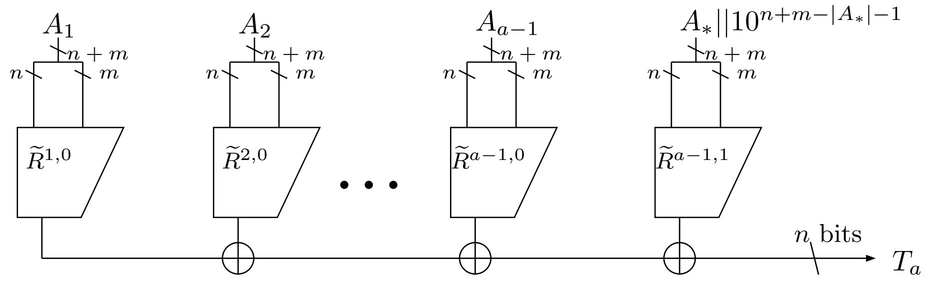
Encrypting a message whose length is a multiple of the block length. No padding is needed.



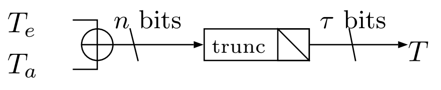
Encrypting a message whose length is not a multiple of the block length. The final message block is padded to make it a full block



Computing T_a for an associated data whose length is a multiple of the input length (i.e $|A_a| = n+m$).



Computing T_a for an associated data whose length is not a multiple of the input length. The final block is padded to make it a full block .



The T is computed as XOR of T_e and T_a truncated to τ bits.

Figure 3.4 – **The $\mathbb{OMD}[\widetilde{R},\tau]$ scheme** using a tweakable random function $\widetilde{R} \leftarrow_\$ \widetilde{\text{Func}}(\mathcal{T}, 2^{m+n}, 2^n)$ (i.e. $\widetilde{R} : \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \rightarrow \{0,1\}^n$).

$A_1 \cdots A_a$ or $A = A_1 \cdots A_{a-1}A_*$ is the associated data, $\mathbb{C} = C\|T$ is the ciphertext where $C = C_1 \cdots C_\ell$ (with $|C_i| = m$ for $1 \leq i \leq \ell$) or $C = C_1 \cdots C_{\ell-1}C_*$ (with $|C_i| = m$ for $1 \leq i \leq \ell - 1$ and $|C_*| < m$), and $T = (T_e \oplus T_a)[n-1 \cdots n-\tau] \in \{0,1\}^\tau$ is the tag. Let $M = M_1 \cdots M_\ell$ or $M = M_1 \cdots M_{\ell-1}M_*$ denote the corresponding putative plaintext, internally computed by the decryption algorithm.

In order to forge successfully, $\mathscr{A}''$ must find the first τ bits of $T = T_e \oplus T_a$ where $T_e = \widetilde{R}^{\langle N,x,y\rangle}(\text{final input})$ and $T_a = \textsc{Hash}_{\widetilde{R}}(A)$. By "final input" we mean $H_\ell\|M_\ell$ or $H_\ell\|M_*\|10^{m-|M_*|-1}$ when $|C| \neq 0$, in which case the final tweak used to generate T_e will be $\langle N,\ell,1\rangle$ or $\langle N,\ell,2\rangle$ respectively (depending on whether the final block is a full block or not); otherwise (i.e. for empty message) the "final input" will be $H_1\|10^{m-1}$ and hence the final tweak used to generate T_e will be $\langle N,1,2\rangle$. In the following, we consider an empty message to have an "incomplete final block". We have the following disjoint cases:

Case 1: $N \notin \{N^1, \cdots N^{q_e}\}$. The adversary has to find a correct T that is the first τ bits of the value $\widetilde{R}^{\langle N,x,y\rangle}(\text{final input}) \oplus T_a$ but has not seen any image under $\widetilde{R}^{\langle N,x,y\rangle}(\cdot)$, hence the probability that the $\mathscr{A}''$ can succeed in doing this is $2^{-\tau}$.

In all the following cases, we will have $N = N^i$ for some $1 \leq i \leq q_e$. We can ignore all but the ith query, as the replies to those queries are computed with nonces $N^{i'} \neq N$, and thus they are independent of the alleged forgery $N, A, \mathbb{C}$.

Case 2: $N = N^i$, $|C| \neq |C^i|$, and one of $|C|$ and $|C^i|$ is a non-zero multiple of m but the other is not. Even if $\mathscr{A}''$ knows T_a, computing the correct T requires guessing τ bits of an image under $\widetilde{R}^{\langle N,x,y\rangle}$, which we show to not have been evaluated throughout the game. Consider the case that $|C^i|$ is non-zero a multiple of m but $|C|$ is not; then $x = \ell$ or $x = 1$ (if $|C| = \ell = 0$) and $y = 2$, so $\mathscr{A}''$ must guess the first τ bits of the value $\widetilde{R}^{\langle N,x,2\rangle}(\text{final input}) \oplus T_a$ but has seen no image under $\widetilde{R}^{\langle N,x,2\rangle}(\cdot)$. In the case when $|C|$ is a non-zero multiple of m but $|C^i|$ is not, $x = \ell$ and $y = 1$ so $\mathscr{A}''$ must guess the first τ bits of the value $\widetilde{R}^{\langle N,\ell,1\rangle}(\text{final input}) \oplus T_a$, but it has seen no image under $\widetilde{R}^{\langle N,\ell,1\rangle}(\cdot)$. Therefore, the probability that the adversary can succeed in guessing T is $2^{-\tau}$.

Case 3: $N = N^i$, $|C| \neq |C^i|$, and either both $|C|$ and $|C^i|$ are non-zero multiples of m or none of them is. If both $|C|$ and $|C^i|$ are non-zero multiples of m then $|C| \neq |C^i|$ means that $\ell \neq \ell^i$, it can be easily seen that in this case even if the adversary knows T_a it must still guess the first τ bits of the value $\widetilde{R}^{\langle N,\ell,1\rangle}(\text{final input})$ (see Figure 3.4) while it has seen no image of this function; the probability to succeed in guessing T is clearly $2^{-\tau}$.

Now, let's consider the case that neither $|C|$ nor $|C^i|$ is a non-zero multiple of m; then $|C| \neq |C^i|$ means that we have three sub-cases: (3a) $\ell = 1$ and $\ell^i = 0$ or vice-versa (3b) other cases when $\ell \neq \ell^i$ and $0 < \ell, \ell^i$, (3c) $\ell = \ell^i$ but $|C_*| \neq |C^i_*|$. We address the case (3a) last.

(3b) It can be seen the adversary must guess the first τ bits of the random function $\widetilde{R}^{\langle N,\ell,2\rangle}$ while has seen no image of this function; the chance to do so is clearly $2^{-\tau}$.

(3c) The adversary must guess the first τ bits of $\widetilde{R}^{\langle N,\ell,2\rangle}(H_*\|(M_*\|10^{m-|M_*|-1}))$ while it has only seen (τ bits of) a single image of this function for one *different* domain point, namely $(H_*^i\|(M_*^i\|10^{m-|M_*^i|-1}))$; the probability to succeed in this case is again $2^{-\tau}$. (Note that $|M_*| = |C_*|$ and $|M_*^i| = |C_*^i|$. Using 10^* padding for processing messages whose length is not a multiple of m is essential for this part.)

(3a) This case is similar to the case (3c); Both T_e and T_e^i are produced by $\widetilde{R}^{N,1,2}$ but necessarily by different inputs, due to the use of injective padding.

Case 4: $N = N^i$, $|C| = |C^i|$, and $A \neq A^i$. We consider two subcases: (4a) where $|A| \neq 0$ and (4b) where $|A| = 0$.

(4a) Let's assume that we even provide $\mathscr{A}''$ with the correct value of T_e which will only make it's job easier. Then the adversary's task will reduce to guessing a correct value for the first τ bits of T_a. The only relevant information that the adversary has is the first τ bits of T_a^i. We show that even if the whole T_a^i is given to the adversary, the chance to correctly guess the first τ bits of T_a is still $2^{-\tau}$. This is done by a simple sub-case analysis:

1. if only one of $|A|$ and $|A^i|$ is a multiple of $n+m$ then it is easy to see (from Figure 3.4) that the probability to guess the first τ bits of T_a is still $2^{-\tau}$;
2. if $a \neq a_i$ then again from Figure 3.4 we can see that the probability to guess the first τ bits of T_a is $2^{-\tau}$;
3. otherwise, we have $a = a_i$ and either both $|A|$ and $|A^i|$ are multiple of $n+m$ or neither of them is a multiple of $n+m$. These two cases are similar. Let's consider the first one. As we have $A \neq A^i$ then it must be the case that for some j we have $A_j \neq A_j^i$. So, the j^{th} value xored to T_a , i.e. $\widetilde{R}^{\langle j,0\rangle}(A_j)$ is a fresh n-bit random value; hence the adversary's chance to guess the first τ bits of T_a is $2^{-\tau}$.

(4b) in order for the forgery attempt $(N, \varepsilon, C\|T)$ to succeed, $\mathscr{A}''$ must find the value of $T = T_e[n-1 \cdots n-\tau]$ produced as $\widetilde{R}^{N^i,\ell^i,j}$. The only image under this function that has been computed in the whole game is T_e^i. However, the adversary has no information about T_e^i, as the distribution of $T^i = T_e^i \oplus T_a^i$ is uniform and independent of T_e^i due to T_a^i that is unknown to $\mathscr{A}''$. So, the probability that the adversary can correctly guess the first τ bits of $T_e = \widetilde{R}^{\langle N,\ell,j\rangle}$(final input) for $j = \{1,2\}$ is $2^{-\tau}$. (Note that $j=1$ when $|C|$ is a multiple of m and $j=2$ when $|C|$ is not a multiple of m).

Case 5: $N = N^i$, $A = A^i$, and $|C| = |C^i| = \ell m$ is a multiple of m. Let's assume that we provide $\mathscr{A}''$ with the correct value $T_a = T_a^i$. We further assume that $C \neq C^i$

as otherwise any $T \neq T^i$ will be incorrect and rejected. Therefore, we may assume that $C_j \neq C_j^i$ for some $1 \leq j \leq \ell$. Now referring to (the top of) Figure 3.4 it is easy to see that if $C_\ell \neq C_\ell^i$ then the probability that the adversary can correctly guess the value of T is $2^{-\tau}$; otherwise there are two cases: (1) if $H_\ell \neq H_\ell^i$ the chance that T is correct is $2^{-\tau}$; (2) if the event $H_\ell = H_\ell^i$ happens then adversary can simply use $T = T^i$. However, for this to occur, there must be a $1 \leq j \leq \ell - 1$ such that $H_j^i \| M_j^i \neq H_j \| M_j$ but $H_{j+1}^i = H_{j+1}$. This happens with probability at most $\ell 2^{-n}$ by union bound, noting that $|H_i| = n$ So, the total success probability in this case is bounded by $\frac{1}{2^\tau} + \frac{\ell}{2^n}$.

Case 6: $N = N^i$, $A = A^i$, and $|C| = |C^i|$ is not a multiple of m. It is easy to see from Figure 3.4 that the analysis of this case is the same as that of Case 5 and the success probability of the adversary is bounded by $\frac{1}{2^\tau} + \frac{\ell}{2^n}$.

Finally, using the result of Bellare et al. [BGM04] that reduces an adversary with multiple forgery attempts to an adversary that has a single forgery attempt, we bound the probability of forgery by adversaries that make q_d decryption (verification) queries by $\frac{q_d}{2^\tau} + \frac{q_d \ell}{2^n}$. □

Instantiating Tweakable RFs with PRFs. We replace the (tweakable) RF $\widetilde{R} \in \widetilde{\text{Func}}(\mathcal{T}, 2^{m+n}, 2^n)$ in $\mathbb{OMD}$ with a (tweakable) PRF $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$. We note that whether we interpret a part of the input to either $\widetilde{R}$ or $\widetilde{F}$ as a tweak or not, the notion of PRF security applies to both situations. The following lemma states the classical bound on the security loss induced by this replacement step.

Lemma 3.3. *Let $\widetilde{R} : \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be a RF and $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be a keyed function. Let $\mathscr{A}$ be a CPA adversary that runs in time t, makes q_e encryption queries that induce no more than σ_e n calls to F in total, such that no individual input (AD or message) is more than ℓ_{max} blocks long. Let further $\mathscr{A}'$ be a CCA adversary that runs in time t', makes q_e' encryption queries and q_d' decryption queries that induce no more than σ' calls to F in total, such that no individual input is more that ℓ_{max}' blocks long. Then*

$$\begin{aligned}
\mathbf{Adv}^{\mathbf{priv}}_{\mathbb{OMD}[\widetilde{F},\tau]}(\mathscr{A}) &\leq \mathbf{Adv}^{\mathbf{priv}}_{\mathbb{OMD}[\widetilde{R},\tau]}(\mathscr{B}) + \mathbf{Adv}^{\mathbf{prf}}_{\widetilde{F}}(\mathscr{C}) \\
\mathbf{Adv}^{\mathbf{auth}}_{\mathbb{OMD}[\widetilde{F},\tau]}(\mathscr{A}') &\leq \mathbf{Adv}^{\mathbf{auth}}_{\mathbb{OMD}[\widetilde{R},\tau]}(\mathscr{B}') + \mathbf{Adv}^{\mathbf{prf}}_{\widetilde{F}}(\mathscr{C}')
\end{aligned}$$

$\mathscr{B}$ and $\mathscr{B}'$ are information theoretic adversaries that have the same resources (except for time complexity) as $\mathscr{A}$ and $\mathscr{A}'$ respectively, and where $\mathscr{C}$ and $\mathscr{C}'$ make no more than σ and σ' queries respectively and run in time $t + \gamma \cdot n \cdot \sigma_e$ and $t' + \gamma' \cdot n \cdot \sigma'$ respectively with some constants γ, γ'.

Proof. We let Π stand for $\mathbb{OMD}$. Starting from the definition of PRIV advantage, we have

$$\begin{aligned}\mathbf{Adv}^{\mathbf{priv}}_{\Pi[\widetilde{R},\tau]}(\mathscr{A}) &= \Pr\left[\mathscr{A}^{\mathbf{priv\text{-}R}_{\Pi[\widetilde{R},\tau]}} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathbf{priv\text{-}I}_{\Pi[\widetilde{R},\tau]}} \Rightarrow 1\right]\\ &= \Pr\left[\mathscr{A}^{\mathbf{priv\text{-}R}_{\Pi[\widetilde{F},\tau]}} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathbf{priv\text{-}R}_{\Pi[\widetilde{R},\tau]}} \Rightarrow 1\right]\\ &\quad + \Pr\left[\mathscr{A}^{\mathbf{priv\text{-}R}_{\Pi[\widetilde{R},\tau]}} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathbf{priv\text{-}I}_{\Pi[\widetilde{F},\tau]}} \Rightarrow 1\right],\end{aligned}$$

where we simply add $0 = \Pr\left[\mathscr{A}^{\mathbf{priv\text{-}R}_{\Pi[\widetilde{R},\tau]}} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathbf{priv\text{-}R}_{\Pi[\widetilde{R},\tau]}} \Rightarrow 1\right]$ to the advantage. We have

$$\mathbf{Adv}^{\mathbf{prf}}_{\widetilde{F}}(\mathscr{C}) \geq \Pr\left[\mathscr{A}^{\mathbf{priv\text{-}R}_{\Pi[\widetilde{F},\tau]}} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathbf{priv\text{-}R}_{\Pi[\widetilde{R},\tau]}} \Rightarrow 1\right]$$

as an adversary $\mathscr{C}$ can be constructed using $\mathscr{A}$ as a subroutine; $\mathscr{C}$ uses its own oracle and follows the code of the algorithms $\mathcal{E}$ and $\mathcal{D}$ to perfectly simulate the Enc and Dec oracles for $\mathscr{A}$. Then $\mathscr{C}$ outputs whatever $\mathscr{A}$ outputs. If $\mathscr{C}$'s oracle implements $\widetilde{F}_K$, then $\mathscr{C}$ perfectly simulates $\mathbf{priv\text{-}R}_{\Pi[\widetilde{F},\tau]}$ for $\mathscr{A}$, while if $\mathscr{C}$'s oracle implements a truly random function $\widetilde{R}$, then $\mathscr{C}$ perfectly simulates $\mathbf{priv\text{-}R}_{\Pi[\widetilde{R},\tau]}$.

We further have that

$$\mathbf{Adv}^{\mathbf{priv}}_{\Pi[\widetilde{R},\tau]}(\mathscr{B}) \geq \Pr\left[\mathscr{A}^{\mathbf{priv\text{-}R}_{\Pi[\widetilde{R},\tau]}} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathbf{priv\text{-}I}_{\Pi[\widetilde{F},\tau]}} \Rightarrow 1\right]$$

as an adversary $\mathscr{B}$ can use $\mathscr{A}$ by simply forwarding all $\mathscr{A}$'s queries and then output whatever $\mathscr{A}$ outputs. Because the games $\mathbf{priv\text{-}I}_{\Pi[\widetilde{F},\tau]}$ and $\mathbf{priv\text{-}I}_{\Pi[\widetilde{R},\tau]}$ are identical, the simulation of games for $\mathscr{A}$ will be perfect. The final bound is obtained by triangle inequality.

A similar transition based on triangular inequality and almost identical reductions can be applied to obtain the bound on AUTH advantage. □

We next instantiate the (tweakable) PRF $\widetilde{F}$ using a PRF F (with a smaller domain) by means of masking a part of the input to F by an offset generated as a function of the key and the tweak, as shown in Fig. 3.5. This method to tweak a PRF is essentially the XE method [Rog04a], originally used to construct tweakable blockciphers. In OMD the tweaks are of the form $\mathsf{T} = (\alpha, i, j)$ where $\alpha \in \mathcal{N} \cup \{\varepsilon\}$, $1 \leq i \leq 2^{n-4}$ and $j \in \{0, 1, 2\}$. We note that not all combinations are used; for example, if $\alpha = \varepsilon$ (empty) which corresponds to processing of the associated data in Figure 3.1, then $j \neq 2$.

Lemma 3.4. *Let $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be a keyed function with key space $\mathcal{K}$. Let $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be defined by $\widetilde{F}^{\langle \mathsf{T} \rangle}_K(X, Y) = F_K((X \oplus \Delta_K(\mathsf{T})), Y)$ for every $\mathsf{T} \in \mathcal{T}, K \in \mathcal{K}$, $X \in \{0,1\}^n, Y \in \{0,1\}^m$, and let $\Delta_K(\mathsf{T})$ be the masking function of OMD as defined in Section 3.4. Let further $\mathscr{A}$ be an adversary that runs in time t and makes q queries. Then we have*

$$\mathbf{Adv}^{\mathbf{prf}}_{\widetilde{F}}(\mathscr{A}) \leq \mathbf{Adv}^{\mathbf{prf}}_{F}(\mathscr{B}) + \frac{3q^2}{2^n}$$

for some $\mathscr{B}$ that runs in time $t + \gamma \cdot q$ for a constant γ and makes $2q$ queries.

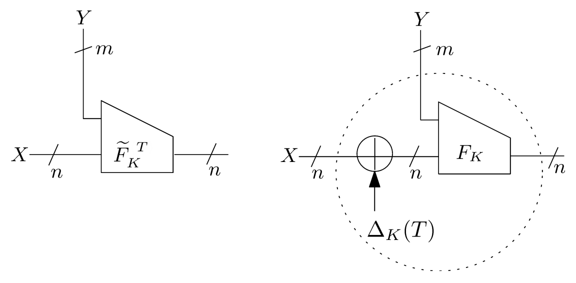


Figure 3.5 – **Constructing a tweakable PRF** $\widetilde{F}_K^{\langle \mathsf{T} \rangle} : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n$ using a PRF $F_K : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n$. There are several efficient ways to define the masking function $\Delta_K(\mathsf{T})$ [Rog04a, CS08, KR11]; we use the method used in OCB3 [KR11].

Proof. The proof is a simple adaptation of a similar result on the security of the XE construction (to tweak a blockcipher) [KR11]. We sketch the main steps of the proof and refer the reader to the original proof [Rog04a, KR11] for details. The key property required for the proof to apply is that the masking function $\Delta_K(\mathsf{T})$ must be a 2^{-n}-uniform and 2^{-n}-AXU hash. This is shown at the end of the proof.

As we use a PRF rather than PRP, our bound has two main terms. The first term is a single birthday bound loss of $\frac{0.5q^2}{2^n}$ to take care of the case that a collision might happen when computing the initial mask $\Delta_{N,0,0} = F_K(N \| 10^{n-1-\nu}, 0^m)$ using a PRF (F) rather than a PRP (as in the original proof [KR11]). The analysis of the remaining term (i.e. $\frac{2.5q^2}{2^n}$) is essentially the same as the corresponding part in the original proof [KR11], but we note that in the context of our construction as we are directly dealing with PRFs (unlike the original analysis [KR11] in which PRPs are used), the bound obtained here does not have any loss terms caused by the switching (RP-RF) lemma. Therefore, instead of the original $\frac{6q^2}{2^n}$ bound [KR11] (from which $\frac{3.5q^2}{2^n}$ is due to using the switching lemma) our bound has only $\frac{2.5q^2}{2^n}$.

We now show that the masking function $\Delta_K(T) = \Delta_K(\alpha, i, j)$ is a 2^{-n}-almost universal 2^{-n}-AXU hash; it outputs an n-bit mask such that the following two properties hold for any fixed string $H \in \{0,1\}^n$:

1. $\Pr[\Delta_K(\alpha, i, j) = H] \leq 2^{-n}$ for any (α, i, j)
2. $\Pr[\Delta_K(\alpha, i, j) \oplus \Delta_K(\alpha', i', j') = H] \leq 2^{-n}$ for $(\alpha, i, j) \neq (\alpha', i', j')$

where the probabilities are taken over random selection of the key.

The masking scheme of OMD is an adaptation of that used in OCB3 [KR11]. It is easy to verify that it satisfies these two properties. Given a $\mathsf{T} = (\alpha, i, j)$, the corresponding mask is always computed as

$$\Delta_K(\mathsf{T}) = \mathsf{N}_\alpha \oplus 2^2 \cdot \gamma_i \cdot L_* \oplus \mathsf{j}_{j,\alpha} \cdot L_* = \mathsf{N}_\alpha \oplus (2^2 \cdot \gamma_i \oplus \mathsf{j}_{j,\alpha}) \cdot L_*$$

where $L_* = F_K(0^n, 0^m)$, γ_i is the ith codeword of the canonical Gray code, $\mathsf{N} = 0^n$ if

$\alpha = \varepsilon$ and $\mathsf{N}_\alpha = F_K(\alpha\|10^{n-1-\nu}, 0^m)$ otherwise, and

$$\mathsf{j}_{j,\alpha} = \begin{cases} 0 & \text{if } j = 0 \\ j+1 & \text{if } \alpha \neq \varepsilon \text{ and } j \in \{1,2\} \\ 1 & \text{if } \alpha = \varepsilon \text{ and } j = 1. \end{cases}$$

The canonical Gray code is defined as $\gamma_0 = 0^n$ and $\gamma_i = \gamma_{i-1} \oplus 2^{\mathtt{ntz}(i)}$ for $i \geq 1$.

The 2^{-n}-almost universal property of the masking scheme follows from the fact that each mask $\Delta_K(\alpha, i, j)$ contains a non-zero multiple of uniformly distributed L_*, and for $\alpha \neq \varepsilon$ the variable N_α is computed by evaluating F_K on a non-zero input.

To verify the 2^{-n}-AXU property, we perform a short case analysis. If $\alpha \neq \alpha'$, then necessarily $\alpha\|10^{n-1-\nu} \neq \alpha'\|10^{n-1-\nu}$ and the two variables N_α and $\mathsf{N}_{\alpha'}$ are both uniformly distributed and independent.

If $\alpha = \alpha'$, then we must have $(i,j) \neq (i',j')$. For the canonical Gray code, we have that for any $i \neq i'$ we have $\gamma_i \neq \gamma_i$, and for any i $0 \leq \mathsf{int}(\gamma_i) \leq 2i$. If $i \leq 2^{n-4}$, then γ_i will have degree smaller or equal to $n-3$ (as a polynomial), so $2^2 \cdot \gamma_i$ will always have the two least significant bits set to 0 (as the corresponding multiplication in $\mathsf{GF}(2^{128})$ will never require a reduction). It follows that if $(i,j) \neq (i',j')$, then necessarily $(2^2 \cdot \gamma_i \oplus \mathsf{j}_{j,\alpha}) \neq (2^2 \cdot \gamma_{i'} \oplus \mathsf{j}_{j',\alpha})$ and thus the collision of masks is impossible in this case. □

3.7 Performance

In this section, we briefly describe the performance of OMD in software. As a mandatory part of the CAESAR submission, both reference and optimized implementations of OMD needed to be presented.

The reference implementation. The reference implementations of OMD-sha256 and OMD-sha512 in C language were co-developed by the author of this thesis and Simon Cogliani. The main objective of the reference implementation was portability and readability of the code, thus both implementations are using a vanilla implementation of their respective SHA2 compression functions from OpenSSL [Fou18]. While not being designed for speed, the reference implementations were used for comparison in the benchmarks of the optimized implementations on various platforms.

Optimized implementation for x86_64 architecture. Because the CPUs by Intel and AMD are the most common CPUs on high-end computers, the x86_64 architecture was a natural target for the first optimized implementations of OMD.

The optimized implementations for this platform were developed by Ralph Ankele and Robin Ankele during a student project [AA14], the implementation part of which was supervised by the author of this thesis. Two of the implementations rely on the extended instruction sets SSE4 and AVX1 [Cor], which both provide SIMD instructions that can be applied to four, or eight 32-bit integers at the same time, respectively (AVX1 is an improvement of SSE4). The third implementation uses the announced SHA-Extension [GGY$^+$13], which provides hardware acceleration for the compression functions of SHA1 and SHA256. Originally announced to be released in 2015 [GGY$^+$13], the extension first appeared in the Goldmont architecture in 2016 [Par16, Cor18].

All three optimized implementations share the same high-level structure. We recall that $\ell_{max} = \max_{X \in \mathcal{A} \cup \mathcal{M}}(|X|_m)$ denotes the upper bound on the maximum number of blocks in any input to the encryption algorithm.

Precomputation. The values L_* and $L[i]$ for $i = 1, \ldots, \lceil \log_2(\ell_{max}) \rceil$ are precomputed only once using a dedicated API call, and stored to be used by every encryption (or decryption) query. This allows to partially amortize the computational cost of the masking at the expense of memory. The required memory will grow logarithmically with ℓ_{max}.

We note that a more aggressive precomputation is also possible, one where we compute $\bar{\Delta}_{i,0}$ for each $i \leq \ell_{max}$. This will further decrease the computational complexity of individual encryption (or decryption) queries, but the required memory will grow linearly with ℓ_{max}. This option was used for the measurements.

Instruction extensions. As the computationally heaviest component of OMD, the speedup of the compression function sha-256 (or sha-512) will have the biggest impact. For this, Ankele and Ankele took advantage of assembler implementations of sha-256 (or sha-512) compression function that uses the advanced instructions of the SSE4/AVX1 [GYG12, GCG12]. They additionally implemented OMD using the SHA instruction extension set, but this implementation was never tested.

Compiler optimizations. In order to discover the maximal potential speed of OMD, all software was compiled with the Ofast optimization flag of gcc. (This is equivalent to the O3 flag with some additional aggressive optimizations for floating point operations).

Implementation tricks. Further measures that improve the performance, albeit marginally, were applied, such as loop unrolling, use of C macros, avoiding type conversions etc.

Ankele and Ankele performed experiments to asses the speedup obtained by the SSE4 and AVX1-based implementations relative to the reference implementation. The SHA extension was not released in time for the optimized implementation to be tested and benchmarked before OMD finished in CAESAR.

The experiments were conducted on a 64-bit 2.4GHz dual-core Intel Core i5-2415M processor running Ubuntu 12.10. All implementations (including the reference implementation) were compiled with the gcc-4.7.2 (Ubuntu/Linaro 4.7.2-2ubuntu1) compiler, using the Ofast flag.

Ankele and Ankele measured the performance of the implementations in computational cycles spent per one byte of input data (the lower the better), such that the length of input data was computed as a sum of the lengths of message and AD in bytes. The time stamp counter of the CPU was read with the RDTSC instruction to compute the total numbers of spent cycles.

The performance of each implementation was measured for messages of length from 128 bits to 4096 bits (with a step of 128 bits), each combined with AD of length from 0 bits to 4096 bits (with a step of 128 bits). To reduce noise during measurements (e.g. from process context switches) Ankele and Ankele used the same method as Krovetz and Rogaway [KR11] in their benchmarking of OCB3: for each measurement, take the median of 91 average numbers of cycles, each obtained using 200 timings.

The comparison of the three implementations with empty AD for both OMD-sha256 and OMD-sha512 can be seen in Figure 3.6. The complete sets of measurements are visualized in Appendix A.1.

We see that both the optimized implementation based on SSE4 and the one based on AVX1 are about twice as fast as the reference, with the one based on AVX1 being slightly faster. The gradual acceleration of the encryption with the increasing message length occurs because there is a single call to the compression function to process the nonce at the beginning of each query; the longer the query the better is this amortized. Overall, the performance of OMD is not bad, however with more than 20 cycles per byte for a message of 4096 bits, it cannot compete with e.g. OCB that performs under 1 cycle/byte.

Optimized implementation for the ARM architecture. We chose ARM as the next target for an optimized implementation, due to its popularity in low-end devices, such as smartphones.

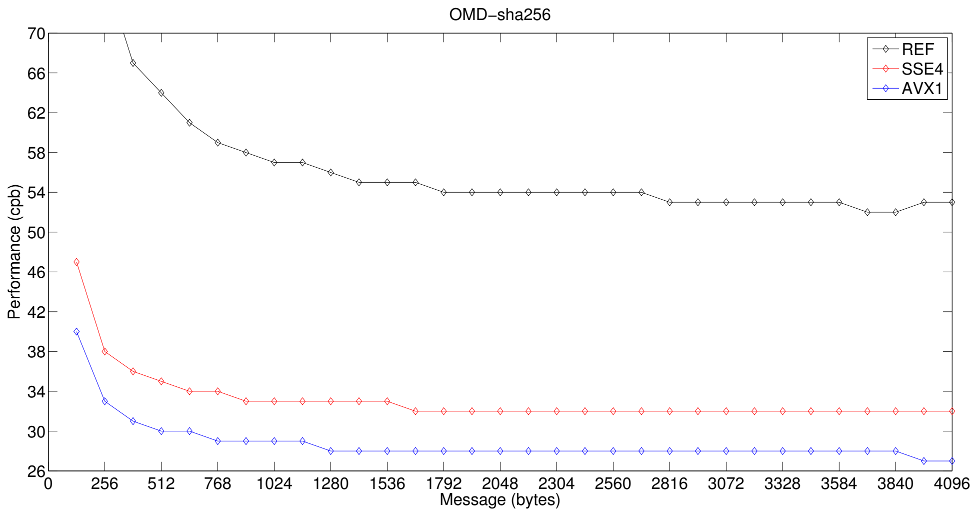
The optimized implementation for this platform was developed by Johan Droz during a student project [Dro15] supervised by the author of this thesis.

The optimized implementation was targeted at 32-bit ARM processors, and was developed and tested on an ARMv7 CPU. We therefore only optimized OMD-sha256. The main ideas of the optimization are similar as for the x86_64 architecture:

Precomputation. Compute the L-values once and store them for use in encryption (and decryption) queries.

Compression function in assembly. Optimize the computationally heaviest part of OMD–the compression function. Targeting older ARM versions with no SIMD instruction extensions, Droz provided a dedicated implementation of sha-256 in ARM assembly without using any special intrinsics.

Droz benchmarked the optimized implementation on a Sony X-Peria M using a similar experimental method as for the x86_64 platform, except this time, for each measurement the minimum of 91 average timings obtained from 200 timings each was taken instead of the mean. The optimized implementation peaked at about 70 cycles/byte for a message



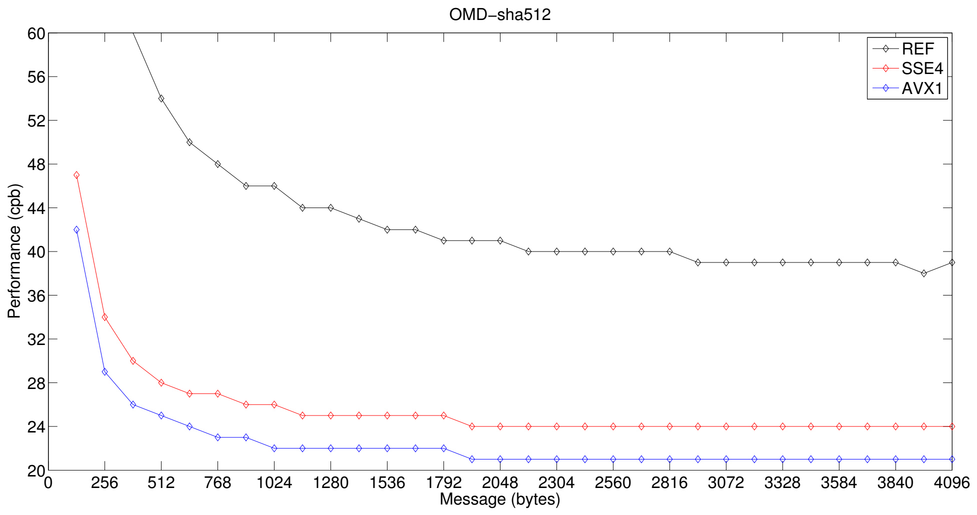


Figure 3.6 – **The comparison of the x86_64-performance** of the reference, the SSE4-based, and the AVX1-based implementations of OMD-sha256 (**top**) and OMD-sha512 (**bottom**) on an Intel Core i5-2415M processor. The performance of the implementations is measured in computational cycles spent per one byte of input data. The AD is empty. The graphs are taken from [AA14]

of 4096 bits. The same implementation with the assembly-based sha-256 replaced by OpenSSL-based version reached about 82 cycles/byte. A comparison is depicted in Figure 3.7.

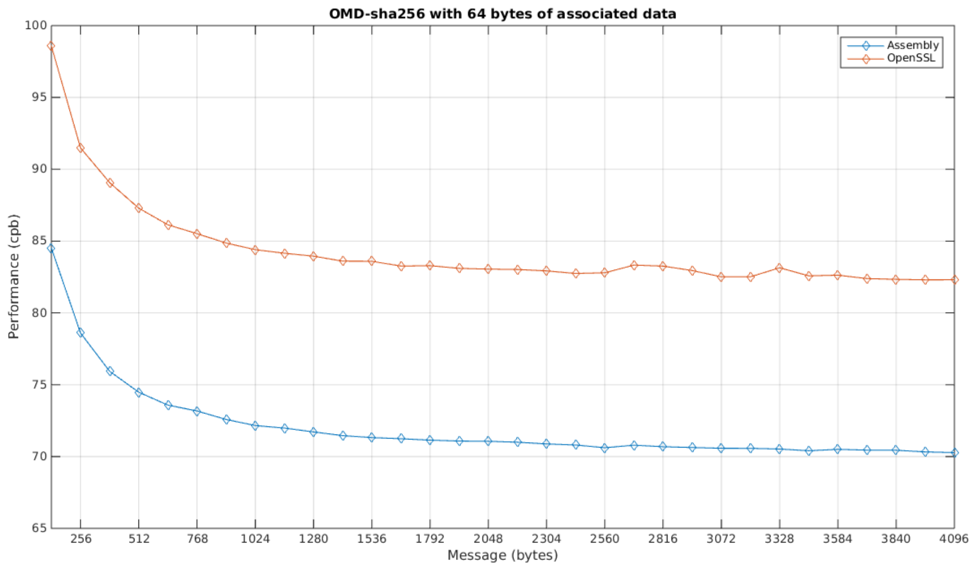


Figure 3.7 – **The comparison of ARM-performance** of the optimized implementation of OMD-sha256 using the assembly implementation of sha-256 versus an implementation of sha-256 from OpenSSL on an ARMv7 processor. Performance is measured in computational cycles spent per one byte of input data. The AD is 64 bytes long. The graph is taken from [Dro15]

Optimized implementation for the MIPS architecture. An optimized implementation of OMD was also developed for the MIPS architecture, as another very common architecture used in embedded systems and networking devices.

The optimized implementation for this platform was developed by Martin Georgiev during a student project [Geo15] supervised by the author of this thesis.

The optimized implementation was developed and tested on a Linksys WRT160NL router equipped with a MIPS 24Kc V7.4 CPU with 32-bit registers. We therefore only optimized OMD-sha256. The main elements of the optimization are the same as for the previous architectures:

Precomputation. Compute the L-values once and store them for use in encryption (and decryption) queries.

Compression function in assembly. Optimize the compression function. Georgiev designed a dedicated implementation of sha-256 in MIPS assembly.

Georgiev benchmarked the optimized implementation using the same experimental method as for the ARM architecture. Because there was no instruction that would allow to count clock cycles, the numbers of consumed cycles were estimated using the wall clock and sufficiently many redundant measurements. The optimized implementation peaked at about 270 cycles/byte for a message of 4096 bits. The same implementation with the assembly-based sha-256 replaced by OpenSSL-based version peaked at about 580 cycles/byte. A comparison is depicted in Figure 3.8.

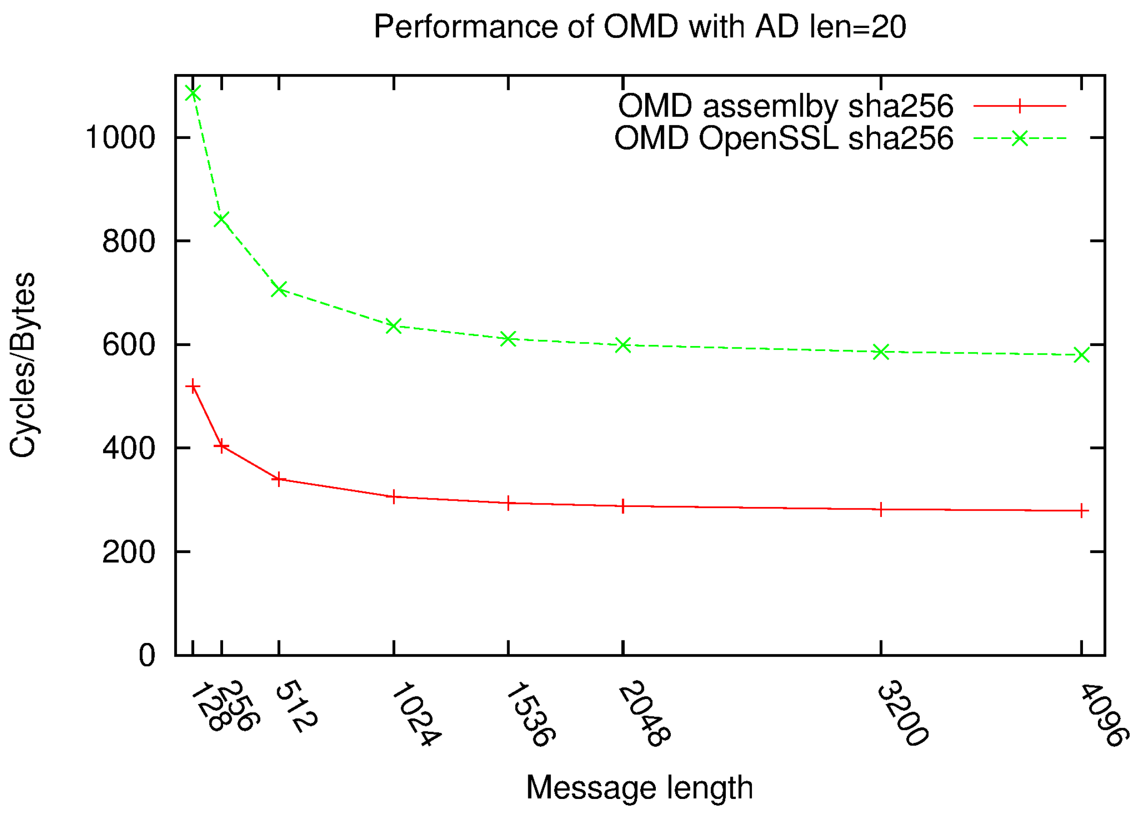


Figure 3.8 – **The comparison of MIPS-performance** of the optimized implementation of OMD-sha256 using the assembly implementation of sha-256 versus an implementation of sha-256 from OpenSSL on a MIPS 24Kc V7.4 processor. The performance of the implementations is measured in computational cycles spent per one byte of input data. The AD is 20 bytes long. The graph is taken from [Geo15]

Comparison to other CAESAR candidates. The software performance of all first and second round CAESAR candidates on several platforms (including x86_64) was carried out by the SUPERCOP framework [lab08]. Only the optimized implementations of OMD for x86_64 were submitted to the SUPERCOP benchmarking before OMD finished in CAESAR. An independent software benchmarking of second round CAESAR candidates on the same platform was done by Ankele and Ankele [AA16].

Based on the results from SUPERCOP [lab18], the optimized implementations of OMD-sah512 peaked around 13 cycles/byte for queries with both message and AD of

2000 bytes, and OMD-sha256 peaked around 17 cycles per byte in the same setting. These results are slight improvements over the observations from the initial measurements of Ankele and Ankele [AA14].

Compared to other candidates, the performance of OMD is not spectacular. The fastest schemes in the competition achieve about 0.3 cycle/byte in ideal conditions. When ordered by speed, OMD would be placed close to the middle of the list of all first round CAESAR candidates. This is also confirmed by the results of Ankele and Ankele [AA16].

The situation would be likely much improved if OMD implemented with the SHA extension was included. Based on simulations performed by Ankele and Ankele [AA14], we conjecture that its throughput would be twice higher than the throughput of the currently fastest implementations of OMD-sha512.

Chapter 4

Misuse-Resistant Variants of OMD

In this chapter, we introduce MR-OMD and PMR-OMD, two nonce-misuse resistant variants of the AE scheme OMD which are, respectively, sequential and fully parallelizable.

The results presented in this chapter come from a joint work with Reza Reyhanitabar and Serge Vaudenay which was published in ProvSec 2014 [RVV14].

Organization of the Chapter. We give a brief overview of the related work in Section 4.1 and a summary of the contribution in Section 4.2.

In Section 4.3, we introduce MR-OMD. We give a description of MR-OMD in Section 4.4, security analysis in Section 4.5 and very briefly discuss a parallelizable variant of MR-OMD in Section 4.6.

4.1 Related Work

The security notion targeted by MR-OMD is the MRAE security by Rogaway and Shrimpton [RS06b] (see Section 2.4). The PRF-then-encrypt paradigm used in MR-OMD is inspired by the SIV construction from the same publication. Unlike SIV, MR-OMD uses a single secret key. Other single-key MRAE-secure schemes preceding MR-OMD are HBS [IY09b] and BTM [IY09a]. Compared to HBS and BTM which use polynomial-based hashing, and need general finite field multiplications in their IV generation part, MR-OMD uses compression function-based hashing and only needs doubling (multiplication by 2) operation in GF(2^n). HBS and BTM also use the PRF-then-encrypt paradigm. This general structure is described as one of the generic composition methods (called "Scheme A4") by Namprempre et al. [NRS14]. There is also another subtle difference between the design of MR-OMD with those of SIV, HBS and BTM; namely, while the latter schemes incorporate the nonce (if used) and the associated data as parts of a vector-based header, our scheme treats the nonce and associated data as different elements. As stated by Rogaway and Shrimpton [RS06b] "the MRAE goal is conceptually different from the DAE goal, the former employing an IV and gaining for this a stronger notion of security. The header and the IV are conceptually different, the one being user-supplied data that the user wants authenticated, the other being a mechanism-supplied value needed to obtain a strong notion of security.”

4.2 Contribution

We present MR-OMD, a nonce-misuse resistant variant of the CAESAR candidate OMD, and to our best knowledge the first compression function-based AE scheme that is nonce-misuse resistant.

As a component of MR-OMD, we propose a new dedicated, compression function-based PRF that efficiently processes a nonce, AD and a message, and is almost-fully parallelizable.

4.3 Misuse-Resistant OMD

OMD is a nonce-based, single-pass mode of operation for authenticated encryption with associated data. To the best of our knowledge, it is the first AE scheme and the only CAESAR candidate that uses a compression function as its lower-level primitive. As such, OMD has some promising features. Among them are provable security in the standard model (based on the well-known PRF assumption on the compression function), high bit-security level (127 bits and 255 bits for OMD-sha256 and OMD-sha512, respectively), the ability to process the inputs in a single pass, and the ability to take advantage of the Intel SHA instructions on Goldmont processors and later. [GGY+13, Par16]

However, the security of OMD fully relies on the assumption that implementations always ensure *correct* use of the nonce, namely that the nonce never gets repeated in the encryption queries. If a repetition of the nonces occurs, security will fully collapse.

Aiming at making OMD robust towards nonce reuse while reusing its components as much as possible, we introduce two variants of OMD, called misuse-resistant OMD (MR-OMD) and parallelizable misuse-resistant OMD (PMR-OMD). We target the maximal possible level of robustness against repeated nonces, the MRAE security, so similar to the previously known schemes in this category (e.g., SIV, HBS and BTM) our constructions are necessarily two-pass. The main goals that motivated the design of MR-OMD are the struggle to have a construction that is very similar to OMD (so that common code and hardware can be reused) and to have an efficient, provably secure MRAE scheme at the same time. The design of PMR-OMD further deviates from OMD, providing a fully parallelizable variant, in contrast with OMD and MR-OMD which both have a serial encryption algorithm.

In MR-OMD and PMR-OMD, the two passes are combined in a way that minimizes the incurred additional cost: using a keyed compression function with $(n+m)$-bit input and n-bit output, for processing a message M with associated data A, MR-OMD and PMR-OMD only need $|M|/(n+m)$ more calls to the compression function compared to OMD, where $|M|$ is the bit length of M. Noticing that the encryption pass in OMD requires $1 + |M|/m$ compression function calls, and considering $m = n$ (as suggested in OMD), the overhead incurred by the second pass in our two-pass variants is about 50% of the encryption time for OMD. We note that the *overhead* is independent of A as it is processed in the same way in both algorithms.

4.4 Description

MR-OMD is a compression function mode of operation for nonce-based AE. It has two parameters, a keyed compression function $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ with the key space $\mathcal{K} = \{0,1\}^k$ and $m \leq n$, and an IV length $\tau < n$, which is the same as the ciphertext expansion.

We let MR-OMD-F denote the MR-OMD mode of operation using a keyed compression function $F_K : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ with $m \leq n$ and an unspecified tag length. We let MR-OMD$[F, \tau]$denote the MR-OMD mode of operation using the keyed compression function F_K and the IV of length τ.

An overview. An instance MR-OMD$[F, \tau]$ has a key space $\mathcal{K} = \{0,1\}^k$, any AD space $\mathcal{A} \subseteq \{0,1\}^{\leq (m+n) \cdot 2^{n-5}}$ and any message space $\mathcal{M} \subseteq \{0,1\}^{\leq m \cdot 2^{n-5}}$. We let $\ell_{max} = \max_{X \in \mathcal{A} \cup \mathcal{M}}(|X|_m)$ denote the upper bound on the maximum number of blocks in any single message or AD. We require that the nonce space $\mathcal{N} = \{0,1\}^\nu$ for an integer $1 \leq \nu < n$.

The encryption algorithm of MR-OMD$[F, \tau]$ takes four input arguments (a secret key K, a nonce N, associated data A, a message M) and outputs a ciphertext $\mathbb{C} = \mathsf{IV}||C \in \{0,1\}^{|M|+\tau}$. The ciphertext consists of an initialization vector (IV) and a core ciphertext. The decryption algorithm of MR-OMD$[F, \tau]$ inputs four arguments (a secret key K, a nonce N, associated data A, a ciphertext $\mathsf{IV}||C$) and either outputs the whole corresponding M at once or an error message $\perp$ in case of an authentication failure.

The encryption algorithm consists of two main components. First, a dedicated PRF computes the IV as an image of all four inputs (K,N,A,M). Then an encryption-only subroutine encrypts M using K, the IV and M. The PRF-component processes AD and the message in blocks of $m+n$ bits, similarly as OMD does with the AD. The encryption-only component is either the message-processing part of OMD itself (in MR-OMD), or counter mode (in PMR-OMD). All calls to the keyed compression function are masked by whitening offsets as in OMD, although we now use a different set of tweaks.

A schematic representation of the encryption algorithm of MR-OMD$[F, \tau]$ is shown in Figure 4.1. The decryption algorithm is very similar to the encryption algorithm, except that the ciphertext is first decrypted using IV from the input and then the IV from input is compared to IV' computed over the nonce, AD and the decrypted message. Figure 4.2 shows the algorithmic description of the encryption and decryption algorithms of MR-OMD$[F, \tau]$.

In the rest of this chapter, we use the following notation. For two strings X, Y s.t. $|X| \geq |Y|$ we let both $X \oplus_{msb} Y$ and $Y \oplus_{msb} X$ denote the xor of X and Y padded on the right with zeroes, i.e. $X \oplus Y \| 0^{|X|-|Y|}$.

Computing the masking values. Before each call to the underlying keyed compression function, we xor a masking offset to the input of F. Compared to OMD, the number of disjoint sets of masks used in MR-OMD is higher. We use the following seven sets of masking values:

- masks $\Delta_{N,i,j}$ for $j \in \{0, \ldots, 5\}$ are used in the IV generation process,
- masks $\bar{\Delta}_{\mathsf{IV},i}$ are used in the encryption (and the decryption) process.

In the following, all multiplications are in $\mathsf{GF}(2^n)$.

Initialization. As in OMD, we compute L_* and an array of L-values as a function of the key. We define $L_* = F_K(0^n, 0^m)$, $L[0] = 2^3 \cdot L_*$, and $L[i] = 2 \cdot L[i-1]$ for $i \geq i$. As before, the L-values can be computed as a part of a one-time initialization and stored in a table.

Masking sequence for IV generation. The masks $\Delta_{N,i,j}$ are computed for every encryption (or decryption) query. We define the two initial N-dependent masks $\Delta_{N,0,0} = F_K(N\|10^{n-1-|N|}, 0^m)$ and $\Delta_{N,0,1} = F_K(N\|10^{n-1-|N|}, 0^m) \oplus L_*$. Then, for $i \geq 1$ and $j, j' \in \{0, \ldots, 5\}$ we let
$\Delta_{N,i,j} = \Delta_{N,i-1,j} \oplus L[\mathtt{ntz}(i)]$, and
$\Delta_{N,i,j} = \Delta_{N,i,j'} \oplus (\langle j \rangle_n \oplus \langle j' \rangle_n) \cdot L_*$.

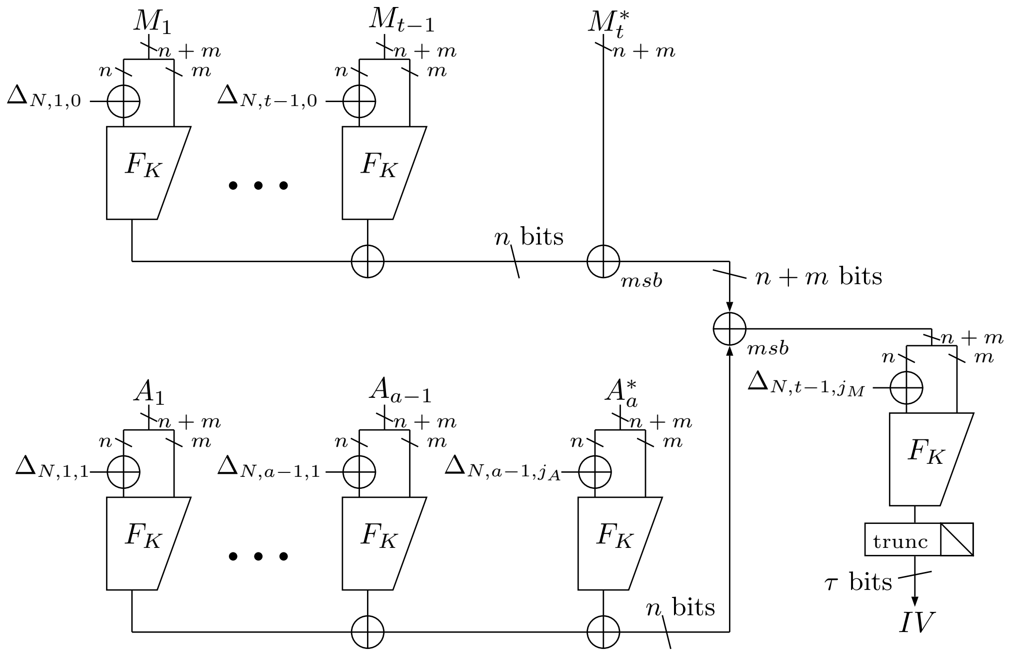
Masking sequence for encryption. We compute the masks $\bar{\Delta}_{\mathsf{IV},i}$ for every query. We define $\bar{\Delta}_{\mathsf{IV},0} = F_K(\mathsf{IV}\|10^{n-1-\tau}, 0^m) \oplus 6 \cdot L_*$, and for $i \geq 1$ we let
$\bar{\Delta}_{\mathsf{IV},i} = \bar{\Delta}_{\mathsf{IV},i-1} \oplus L[\mathtt{ntz}(i)]$.

4.5 Security Analysis

We analyse the security of MR-OMD in two cases: (1) as a MRAE, considering adversaries that are nonce-reusing; (2) in the case that adversaries are nonce-respecting. As MR-OMD is designed as a nonce-misuse resistant scheme, we first focus on analysing the security bounds in the nonce-misuse scenario. The corresponding result is stated in Theorem 4.1. Clearly, an upper-bound for the MRAE advantage also upper-bounds the NAE advantage of adversaries with the same resources. Intuitively, the latter could be lower than the former. This is confirmed by Theorem 4.7.

IV-Based Encryption Schemes. We need to introduce so-called IV-based encryption schemes for the analysis of MR-OMD. The formalism is taken from Rogaway and Shrimpton [RS06b].

An IV-based encryption scheme is a privacy-only scheme. An example of such a scheme can be the CBC mode. Formally, an IV-based encryption scheme is a triplet



If $|M_t| < n + m$ set $M_t^* = M_t||10^{n+m-1-|M_t|}$ and $j_M = 4$. Otherwise $M_t^* = M_t$, $j_M = 2$.
If $|A_a| < n + m$ set $A_a^* = A_a||10^{n+m-1-|A_a|}$ and $j_A = 5$. Otherwise $A_a^* = A_a$, $j_A = 3$.

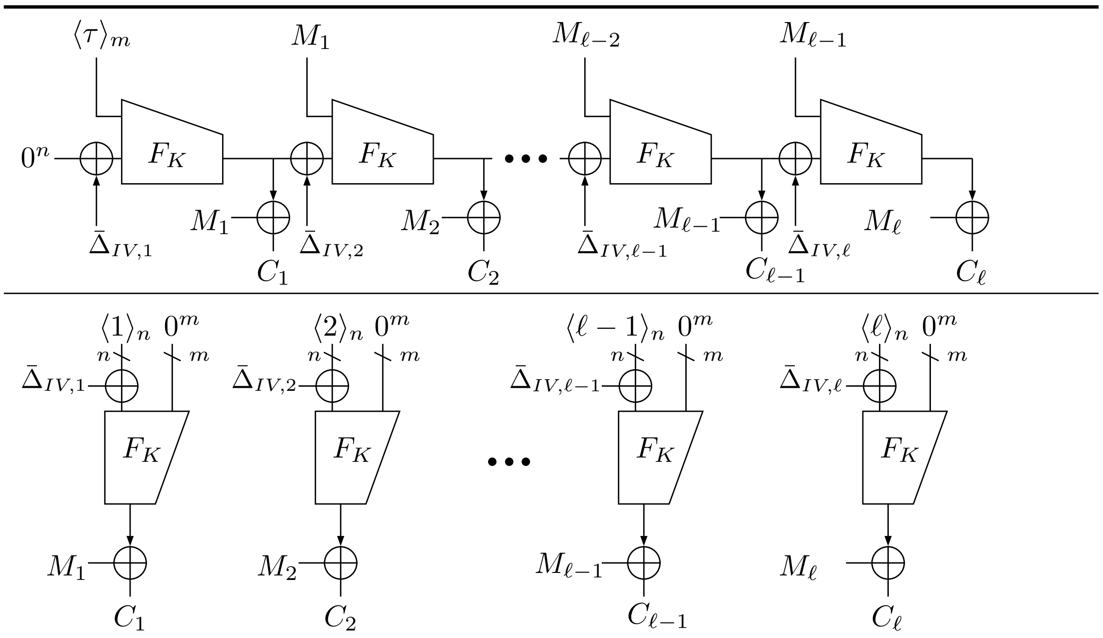


Figure 4.1 – **The encryption algorithm of MR-OMD$[F, \tau]$ and PMR-OMD$[F, \tau]$** using a keyed compression function $F_K : (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ with $m \leq n$. (**Top**) The process of generating the IV. (**Bottom**) The encryption process (upper part for MR-OMD and lower for PMR-OMD). For operation $\oplus_{msb}$ see our convention in Section 4.4.

```
algorithm Initialize($K$)
    $L_* \leftarrow F_K(0^n, 0^m)$
    $L_*^{(2)} \leftarrow 2 \cdot L_*$
    $L_*^{(4)} \leftarrow 2 \cdot L_*^{(2)}$
    $L_*^{(6)} \leftarrow L_*^{(4)} \oplus L_*^{(2)}$
    $L[0] \leftarrow 2 \cdot L_*^{(4)}$
    for $i \leftarrow 1, 2, \cdots$ do
        $L[i] = 2 \cdot L[i-1]$
    end for
    return
end algorithm
```

```
algorithm $\mathrm{HASH}_K(N, A, M)$
    $b \leftarrow n + m$
    $A_1||A_2 \cdots A_{a-1}||A_a \overset{b}{\leftarrow} A$
    $M_1||M_2 \cdots M_{t-1}||M_t \overset{b}{\leftarrow} M$
    $\Sigma_A \leftarrow 0^n$; $\Sigma_M \leftarrow 0^n$
    $\Delta_M \leftarrow F_K(N||10^{n-1-|N|}, 0^m)$
    $\Delta_A \leftarrow \Delta_M \oplus L_*$
    for $i \leftarrow 1$ to $a - 1$ do
        $\Delta_A \leftarrow \Delta_A \oplus L[\mathtt{ntz}(i)]$
        $\mathsf{Left} \leftarrow A_i[b-1 \cdots m]$
        $\mathsf{Right} \leftarrow A_i[m-1 \cdots 0]$
        $\Sigma_A \leftarrow \Sigma_A \oplus F_K(\mathsf{Left} \oplus \Delta_A, \mathsf{Right})$
    end for
    if $|A_a| = b$ then
        $\Delta_A \leftarrow \Delta_A \oplus L_*^{(2)}$
        $\mathsf{Left} \leftarrow A_a[b-1 \cdots m]$
        $\mathsf{Right} \leftarrow A_a[m-1 \cdots 0]$
        $\Sigma_A \leftarrow \Sigma_A \oplus F_K(\mathsf{Left} \oplus \Delta, \mathsf{Right})$
    else if $|A| > 0$ then
        $\Delta_A \leftarrow \Delta_A \oplus L_*^{(4)}$
        $A_a^* \leftarrow A_a||10^{b-|A_a|-1}$
        $\mathsf{Left}A_a^*[b-1 \cdots m]$
        $\mathsf{Right} \leftarrow A_a^*[m-1 \cdots 0]$
        $\Sigma_A \leftarrow \Sigma_A \oplus F_K(\mathsf{Left} \oplus \Delta_A, \mathsf{Right})$
    end if
    for $i \leftarrow 1$ to $t - 1$ do
        $\Delta_M \leftarrow \Delta_M \oplus L[\mathtt{ntz}(i)]$
        $\mathsf{Left} \leftarrow M_i[b-1 \cdots m]$
        $\mathsf{Right} \leftarrow M_i[m-1 \cdots 0]$
        $\Sigma_M \leftarrow \Sigma_M \oplus F_K(\mathsf{Left} \oplus \Delta_M, \mathsf{Right})$
    end for
    if $|M_t| = b$ then
        $\Delta_M \leftarrow \Delta_M \oplus L_*^{(2)}$
        $\mathsf{Left} \leftarrow M_t[b-1 \cdots m] \oplus \Sigma_A$
        $\mathsf{Right} \leftarrow M_t[m-1 \cdots 0]$
        $\mathsf{IV} \leftarrow F_K(\mathsf{Left} \oplus \Delta_M, \mathsf{Right})$
    else
        $\Delta_M \leftarrow \Delta_M \oplus L_*^{(4)}$
        $M_t^* \leftarrow M_t||10^{b-|M_t|-1}$
        $\mathsf{Left} \leftarrow M_t^*[b-1 \cdots m] \oplus \Sigma_A$
        $\mathsf{Right} \leftarrow M_t^*[m-1 \cdots 0]$
        $\mathsf{IV} \leftarrow F_K(\mathsf{Left} \oplus \Delta_M, \mathsf{Right})$
    end if
    return $\mathsf{IV}[n-1 \cdots n-\tau]$
end algorithm
```

```
algorithm $\mathcal{E}_K(N, A, M)$
    if $|N| > n - 1$ then
        return $\perp$
    end if
    $M_1||M_2 \cdots M_{\ell-1}||M_\ell \overset{m}{\leftarrow} M$
    $\mathsf{IV} \leftarrow \mathrm{HASH}_K(N, A, M)$
    $\Delta \leftarrow F_K(IV||10^{n-1-\tau}, 0^m)$
    $\Delta \leftarrow \Delta \oplus L[0] \oplus L_*^{(6)}$
    $H \leftarrow 0^n$
    $H \leftarrow F_K(H \oplus \Delta, \langle\tau\rangle_m)$
    for $i \leftarrow 1$ to $\ell - 1$ do
        $C_i \leftarrow H \oplus M_i$
        $\Delta \leftarrow \Delta \oplus L[\mathtt{ntz}(i+1)]$
        $H \leftarrow F_K(H \oplus \Delta, M_i)$
    end for
    $C_\ell \leftarrow H \oplus M_\ell$
    $\mathbb{C} \leftarrow \mathsf{IV}||C_1||C_2|| \cdots ||C_\ell$
    return $\mathbb{C}$
end algorithm
```

```
algorithm $\mathcal{D}_K(N, A, \mathbb{C})$
    if $|N| > n - 1$ or $|\mathbb{C}| < \tau$ then
        return $\perp$
    end if
    $\mathsf{IV}||C_1||C_2 \cdots C_{\ell-1}||C_\ell \overset{m}{\leftarrow} \mathbb{C}$
    $H \leftarrow 0^n$
    $\Delta \leftarrow F_K(IV||10^{n-1-\tau}, 0^m)$
    $\Delta \leftarrow \Delta \oplus L[0] \oplus L_*^{(6)}$
    $H \leftarrow F_K(H \oplus \Delta, \langle\tau\rangle_m)$
    for $i \leftarrow 1$ to $\ell - 1$ do
        $M_i \leftarrow H \oplus C_i$
        $\Delta \leftarrow \Delta \oplus L[\mathtt{ntz}(i+1)]$
        $H \leftarrow F_K(H \oplus \Delta, M_i)$
    end for
    $M_\ell \leftarrow H \oplus C_\ell$
    $\mathsf{IV}' \leftarrow \mathrm{HASH}_K(N, A, M)$
    if $\mathsf{IV}' = \mathsf{IV}$ then
        return $M \leftarrow M_1||M_2|| \cdots ||M_\ell$
    else
        return $\perp$
    end if
end algorithm
```

Figure 4.2 – **Definition of MR-OMD**$[F, \tau]$ with a keyed compression function $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ s.t. $m \leq n$ and a fixed IV length τ.

$\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ that consists of a key space $\mathcal{K}$ and two "efficient" deterministic algorithms, where the encryption algorithm $\mathcal{E}$ takes a tuple (K, IV, M) as input, such that $K \in \mathcal{K}$, $\mathsf{IV} \in \{0,1\}^\tau$ for some fixed positive τ and $M \in \{0,1\}^*$. We call IV the initialization vector. The notations $\mathcal{E}(K, \mathsf{IV}, M)$, $\mathcal{E}_K(\mathsf{IV}, M)$ and $\mathcal{E}_K^{\mathsf{IV}}(M)$ are used interchangeably. We also assume that if $\mathbb{C} = \mathcal{E}_K^{\mathsf{IV}}(M)$, then we have $|\mathbb{C}| = |M| + \tau$ and $\mathbb{C} = \mathsf{IV}||C$; i.e. the ciphertext reveals IV.

We define the advantage of an adversary $\mathscr{A}$ in breaking the \$-privacy of Π as

$$\mathbf{Adv}_{\Pi}^{\mathbf{priv\$}}(\mathscr{A}) = \Pr\left[K \leftarrow\$ \mathcal{K} : \mathscr{A}^{\mathcal{E}_K^{\$}(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\$(\cdot)} \Rightarrow 1\right]$$

with $\$(\cdot)$ being a random string oracle that on input M returns a random string of length $|M| + \tau$ and $\mathcal{E}_K^{\$}$ returning $\mathcal{E}_K^{\mathsf{IV}}$ with $\mathsf{IV} \leftarrow\$ \{0,1\}^\tau$. It is assumed, that the adversary never asks a query outside the proper message space of Π. Note that in the PRIV\$ security game, the IV is chosen by the game.

Security in the case of nonce misuse. Theorem 4.1 states the MRAE security of MR-OMD. The high-level structure of the proof is similar to the analyses of previous MRAE schemes that follow the synthetic-IV (SIV) design paradigm [RS06b], such as HBS [IY09b] and BTM [IY09a], but the details differ. We first prove the security in the information-theoretic setting using (tweakable) random functions. To obtain the information-theoretic security, we prove security of MR-OMD.HASH as a PRF and the security of MR-OMD.$\mathcal{E}$ as a secure IV-based encryption scheme. Consequently, we prove security of MR-OMD in the MRAE sense using the previous two results. A complexity-theoretic security bound is then determined by instantiating the (tweakable) random functions using the XE construction [Rog04a].

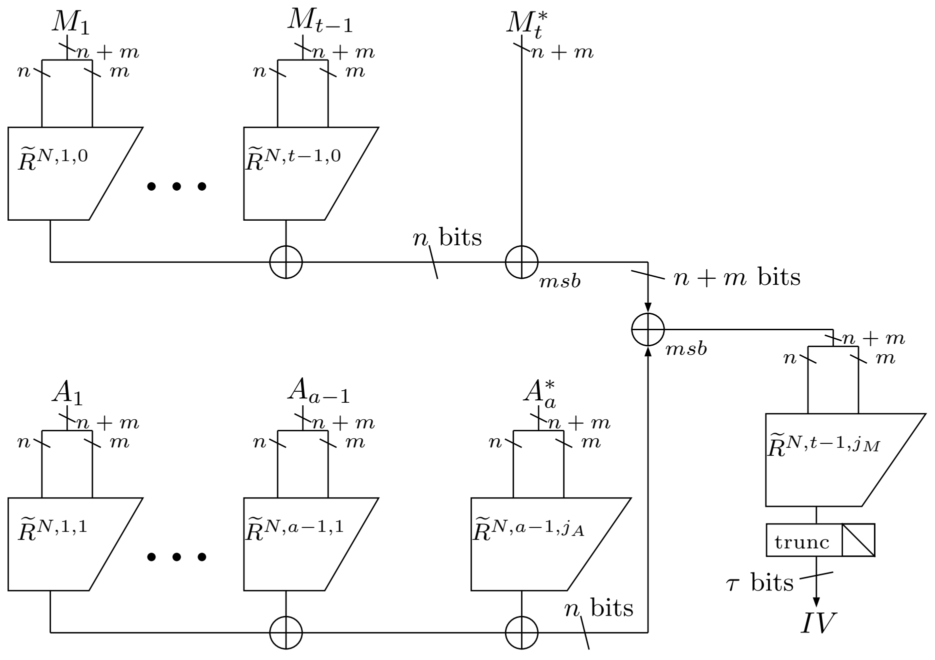
Theorem 4.1. *Fix $n \geq 1$ and $\tau \in \{0, 1, \cdots, n\}$. Let $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be a keyed function, where $1 \leq m \leq n$. Let $\mathscr{A}$ be a CCA adversary that runs in time t, and makes q_e encryption queries and q_d decryption queries that induce no more than σ calls to F in total. Then*

$$\mathbf{Adv}_{MR\text{-}OMD[F,\tau]}^{\mathbf{mrae}}(\mathscr{A}) \leq \mathbf{Adv}_F^{\mathbf{prf}}(\mathscr{B}) + \frac{3.5\sigma^2}{2^n} + \frac{0.5q_e^2}{2^\tau} + \frac{q_d}{2^\tau}$$

for some $\mathscr{B}$ that runs in time $t + \gamma \cdot n \cdot \sigma$ for some constant γ and makes $2 \cdot \sigma$ queries.

Proof. The proof is obtained by combing Lemma 4.4, Lemma 4.2 and Lemma 4.3 with Lemma 4.5 and Lemma 4.6. □

Generalization of MR-OMD based on (tweakable) random functions. We define the scheme $\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R}, \tau]$, a generalization of MR-OMD$[F, \tau]$ that uses a (tweakable) random function $\widetilde{R} \leftarrow\$ \widetilde{\mathrm{Func}}(\mathcal{T}, 2^{m+n}, 2^n)$ instead of the masked function F, as depicted in figure 4.3. The tweak space $\mathcal{T}$ consists of seven mutually exclusive sets of tweaks; namely, $\mathcal{T} = \mathcal{N} \times \mathbb{N} \times \{0\} \cup \mathcal{N} \times \mathbb{N} \times \{1\} \cup \mathcal{N} \times \mathbb{N} \times \{2\} \cup \mathcal{N} \times \mathbb{N} \times \{3\} \cup$



If $|M_t| < n+m$ set $M_t^* = M_t||10^{n+m-1-|M_t|}$ and $j_M = 4$. Otherwise $M_t^* = M_t$, $j_M = 2$.
If $|A_a| < n+m$ set $A_a^* = A_a||10^{n+m-1-|A_a|}$ and $j_A = 5$. Otherwise $A_a^* = A_a$, $j_A = 3$.

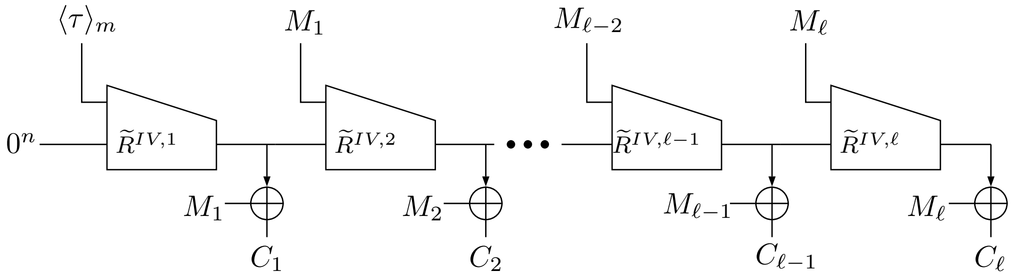


Figure 4.3 – **The scheme $\mathbb{MR}$-$\mathbb{OMD}[\widetilde{\boldsymbol{R}}, \boldsymbol{\tau}]$** using a (tweakable) random function $\widetilde{R} \leftarrow_\$ \widetilde{\mathrm{Func}}(\mathcal{T}, 2^{m+n}, 2^n)$. (**Top**) The process of generating the IV. (**Bottom**) The encryption process. For the operation $\oplus_{msb}$ see our convention in Section 4.4.

$\mathcal{N} \times \mathbb{N} \times \{4\} \cup \mathcal{N} \times \mathbb{N} \times \{5\} \cup \mathcal{IV} \times \mathbb{N}$, where $\mathcal{N} = \{0,1\}^\nu$ is the nonce space and $\mathcal{IV} = \{0,1\}^\tau$ is the set of IV-s.

Lemma 4.2. *Let $\mathbb{MR}$-$\mathbb{OMD}$ $[\widetilde{R}, \tau]$ be the MR-OMD scheme that uses tweakable RF $\widetilde{R}$. Let $\mathscr{A}$ be an adversary that makes no more than q queries that induce no more than σ calls to $\widetilde{R}$ in total. Then*

$$\mathbf{Adv}^{\mathbf{prf}}_{\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R},\tau].\mathrm{HASH}}(\mathscr{A}) \leq \frac{0.5\sigma^2}{2^n}.$$

Proof. We assume w.l.o.g. that the adversary does not repeat a query. Let q denote the number of queries asked by the adversary, and let r denote the number of distinct nonces among all the nonces in the q queries. We partition the queries into sets $\mathcal{Q}^1, \ldots, \mathcal{Q}^r$, so that for any two queries $N, A, M \in \mathcal{Q}^i$ and $N', A', M' \in \mathcal{Q}^j$ we have $N = N'$ if $i = j$ and $N \neq N'$ otherwise. Let $q_i = |\mathcal{Q}^i|$ for $i = 1, \ldots, r$, then we have $q = \sum_{i=1}^{r} q_i$. For any $1 \leq i \leq r$, we will denote the queries from $\mathcal{Q}^i$ as $\mathcal{Q}^i = \{(N^i, A^{i,1}, M^{i,1}), \ldots, (N^i, A^{i,q_i}, M^{i,q_i})\}$. We note that this means, that the adversary uses exactly r distinct nonces in all its queries.

Let $b = n+m$. We will use notation $H^{\tau}_{\widetilde{R}}$ instead of $\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R}, \tau]$.HASH throughout the proof. First, we note that the HASH algorithm can be expressed as

$$H^{\tau}_{\widetilde{R}}(N, A, M) = \widetilde{R}^{\mathsf{T}_{\text{final}}}(h_{\widetilde{R}}(N, A, M))$$

with $T_{\text{final}} \in \mathcal{T}$ and where the function $h_{\widetilde{R}} \in \text{Func}(\mathcal{N} \times \mathcal{A} \times \mathcal{M}, n)$ is defined as

$$h_{\widetilde{R}}(N, A, M) \mapsto \Big(\widetilde{R}^{N,1,1}(A_1) \oplus \ldots \oplus \widetilde{R}^{N,a-1,1}(A_{a-1}) \oplus \widetilde{R}^{N,a-1,j_A}(A^*_a) \oplus$$
$$\widetilde{R}^{N,1,0}(M_1) \oplus \ldots \oplus \widetilde{R}^{N,t-1,0}(M_{t-1})\Big) \oplus_{msb} M^*_t.$$

We claim that

$$\mathbf{Adv}^{\mathbf{prf}}_{H^{\tau}_{\widetilde{R}}}(\mathscr{A}) \leq \max\left\{\sum_{h=1}^{r} \sum_{1 \leq i < j \leq q_h} \Pr\left[\mathsf{coll}\left(\left(N^h, A^{h,i}, M^{h,i}\right), \left(N^h, A^{h,j}, M^{h,j}\right)\right)\right]\right\}$$

with the maximum taken over the choice of $r, q_1, q_2, \ldots, q_r$ and the choice of the queries $(N^{1,1}, A^{1,1}, M^{1,1}), \ldots, (N^{r,q_r}, A^{r,q_r}, M^{r,q_r})$ such that their total length in blocks is limited by σ, i.e. $\sum_{i=1}^{r} \sum_{j=1}^{q_i} \left(|A^{i,j}|_b + \max\left(1, |M^{i,j}|_b\right)\right) \leq \sigma$. We let

$$\mathsf{coll}\left(\left(N^h, A^{h,i}, M^{h,i}\right), \left(N^h, A^{h,j}, M^{h,j}\right)\right)$$

denote the event that $h_{\widetilde{R}}\left(N^h, A^{h,i}, M^{h,i}\right) = h_{\widetilde{R}}\left(N^h, A^{h,j}, M^{h,j}\right)$ and $|M| \equiv 0 \pmod{b}$ iff $|M'| \equiv 0 \pmod{b}$.

In other words, we claim that the advantage $\mathbf{Adv}^{\mathbf{prf}}_{H^{\tau}_{\widetilde{R}}}(\mathscr{A})$ is bounded by the probability of collision on the input to the final (tweakable) RF between two queries with the same nonce (we apply the union bound to obtain the inequality) such that their messages both have a complete final block, or none of them has.

The final call to the (tweakable) RF $\widetilde{R}^{\mathsf{T}_{\text{final}}}$ is independent from $h_{\widetilde{R}}$, because its tweak T_{final} is not used anywhere in $h_{\widetilde{R}}$. Moreover, the final tweaks $\mathsf{T}_{\text{final}}$ and $\mathsf{T}'_{\text{final}}$ of two queries $(N, A, M), (N', A', M')$ will be distinct if $N \neq N'$ or if $|M| \equiv 0 \pmod{b} \not\Leftrightarrow |M'| \equiv 0 \pmod{b}$. Therefore, unless there is a collision on the output of $h_{\widetilde{R}}$ among queries that share the same nonce and have $|M| \equiv 0 \pmod{b} \Leftrightarrow |M'| \equiv 0 \pmod{b}$, the construction $\widetilde{R}^{\{T_{\text{final}}\}}(h_{\widetilde{R}}(N, A, M))$ behaves as a truly RF and cannot be distinguished from such. This completes the proof of the claim.

Now, we proceed to bound the collision probability $\Pr\left[h_{\widetilde{R}}(N, A, M) = h_{\widetilde{R}}(N, A', M')\right]$ for two fixed but arbitrary queries (N, A, M) and (NA', M') by a case analysis. We let $a = |A|_b$, $t = \max(1, |M|_b)$, $a' = |A'|_b$, $t' = \max(1, |M'|_b)$.

Case 1: $|M|$ and $|M'|$ are both multiples of b and (w.l.o.g) $|A|$ is a multiple of b and $|A'|$ is not. The collision on $h_{\widetilde{R}}$ occurs, if $h_{\widetilde{R}}(N, A, M) = h_{\widetilde{R}}(N, A', M')$. Let $S^{(1)}(N, A, M) = h_{\widetilde{R}}(N, A, M) \oplus \widetilde{R}^{N,a-1,j_A}(A_a)$ be a partial result of evaluating $h_{\widetilde{R}}(N, A, M)$. Given any two queries $(N, A, M), (N, A', M')$, a collision on $h_{\widetilde{R}}$ is then equivalent to

$$\begin{aligned} S^{(1)}(N, A, M) \oplus \widetilde{R}^{N,a-1,3}(A_a) =& S^{(1)}(N, A', M') \oplus \widetilde{R}^{N,a'-1,5}(A'_{a'}) \\ \widetilde{R}^{N,a'-1,5}(A'_{a'}) \oplus \widetilde{R}^{N,a-1,3}(A_a) =& S^{(1)}(N, A', M') \oplus S^{(1)}(N, A, M) \end{aligned}$$

The two tweaks used to process the last blocks of A and A' come from mutually exclusive sets, so the images produced from these two blocks will always be uniform and independent. The probability of collision is then $1/2^n$.

Case 2: $|M|, |M'|$ are both multiples of b and $|A|, |A'|$ are either both multiples of b, or they both are not. In case that both A and A' have an incomplete final block, we can assume, that $A \leftarrow A||10^{b-1-|A| \bmod b}$ and $A' \leftarrow A'||10^{b-1-|A'| \bmod b}$. This does not affect the probability of collision because the padding is injective, and because the set of tweaks used to process final blocks of AD with full-length final block ($j_A = 3$) is mutually exclusive with the set of tweaks used to process final block of messages with incomplete final block ($j_A = 5$). Thus in the following sub-cases we can w.l.o.g. assume that $|A|, |A'|$ are multiples of b.

Case 2a: $a \neq a'$. W.l.o.g assume that $a > a'$. Similarly as in **Case 1**, we define a partial evaluation of $h_{\widetilde{R}}(A, M)$ that stops after the first a' blocks of A as follows:

$$\begin{aligned} S^{(2a)}(N, A, M) \mapsto \Big(& \widetilde{R}^{N,1,1}(A_1) \oplus \ldots \oplus \widetilde{R}^{N,a',1}(A_{a'}) \oplus \\ & \widetilde{R}^{N,1,0}(M_1) \oplus \ldots \oplus \widetilde{R}^{N,t-1,0}(M_{t-1})\Big) \oplus_{msb} M_t^*. \end{aligned}$$

The collision occurs if $\widetilde{R}^{N,a'+1,1}(A_{a'+1}) \oplus \ldots \oplus \widetilde{R}^{N,a-1,j_A}(A_a) = h_{\widetilde{R}}(N, A', M')$ $\oplus\ S^{(2a)}(N, A, M)$. Again, this happens if a xor of outputs of multiple independent RFs equals to a distinct value. The probability of finding a tuple of RFs' inputs producing this equality is $1/2^n$.

Case 2b: $a = a'$ and $A \neq A'$. Because $A \neq A'$, there must be an i, s.t. $1 \leq i \leq a$ and $A_i \neq A'_i$. Again, we define a partial evaluation $S^{(2b)}(N, A, M) = h_{\widetilde{R}}(N, A, M) \oplus \widetilde{R}^{N,i,j_i}(A_i)$. The collision occurs if $\widetilde{R}^{N,i,j_i}(A_i) \oplus \widetilde{R}^{N,i,j_i}(A'_i) = S^{(2b)}(N, A, M) \oplus S^{(2b)}(N, A', M')$. In other words, we have a collision if the result of $\widetilde{R}^{N,i,j_i}(A_i) \oplus \widetilde{R}^{N,i,j_i}(A'_i)$ equals to a distinct value. Because $A_i \neq A'_i$, the probability of this event is $1/2^n$.

Case 2c: $A = A'$ and $t \neq t'$. W.l.o.g. assume that $t > t'$. Similarly as in **Case 2a**, we let $S^{(2c)}(N, A, M)$ be the partial result of $h_{\widetilde{R}}(N, A, M)$ that stops after the first t' blocks of M:

$$S^{(2c)}(N,A,M) \mapsto \Big(\widetilde{R}^{N,1,1}(A_1) \oplus \ldots \oplus \widetilde{R}^{N,a-1,j_A}(A_a) \oplus \widetilde{R}^{N,1,0}(M_1) \oplus \ldots \oplus \widetilde{R}^{N,t',0}(M_{t'})\Big) \oplus_{msb} M_t^*.$$

If the collision occurs, we have $\widetilde{R}^{N,t'+1,0}(M_{t'+1}) \oplus \ldots \oplus \widetilde{R}^{N,t-1,0}(M_{t-1}) = S^{(2c)}(N,A,M) \oplus h_{\widetilde{R}}(N,A',M')$. The probability of this event is $1/2^n$.

Case 2d: $A = A'$, $t = t'$ and M, M' differ in blocks with index i, $i < t$. We let $S^{(2d)}(N,A,M) = h_{\widetilde{R}}(N,A,M) \oplus \widetilde{R}^{N,i,0}(M_i)$. Whenever the collision occurs we have $\widetilde{R}^{N,i,0}(M_i) \oplus \widetilde{R}^{N,i,0}(M_i') = S^{(2b)}(N,A,M) \oplus S^{(2b)}(N,A',M')$. By a similar argument as in **Case 2b**, the probability of this collision is $1/2^n$.

Case 2e: $A = A'$, $t = t'$ and M, M' differ only in the last block. Then we have $h_{\widetilde{R}}(N,A,M) \neq h_{\widetilde{R}}(N,A',M')$, so the probability of collision is 0.

Case 3: $|M|$ and $|M'|$ are not multiples of b and (w.l.o.g) $|A|$ is a multiple of b and $|A'|$ is not. This case is analogous to **Case 1**, the only difference is that both M_t and M_t' will be padded before processing (which is of no consequence). By similar argument as in **Case 1**, we conclude that probability of collision is $1/2^n$.

Case 4: $|M|, |M'|$ are not multiples of b and $|A|, |A'|$ are either both multiples of b, or they both are not. In case that both A and A' have an incomplete final block, we can assume, that $A \leftarrow A||10^{b-1-|A| \bmod b}$ and $A' \leftarrow A'||10^{b-1-|A'| \bmod b}$. This does not affect the probability of collision (by the same argument as in **Case 2**). Thus in the following sub-cases, we can assume that $|A|, |A'|$ are multiples of b. As in previous case, we can also let $M \leftarrow M||10^{b-1-|M| \bmod b}$ and $M' \leftarrow M'||10^{b-1-|M'| \bmod b}$. This effectively transforms this case into **Case 2**. We therefore list all the sub-cases only briefly.

Case 4a: $a \neq a'$. The probability of collision is $1/2^n$, similarly as in **Case 2a** .

Case 4b: $a = a'$ and $A \neq A'$. The probability of collision is $1/2^n$, similarly as in **Case 2b** .

Case 4c: $A = A'$ and $t \neq t'$. The probability of collision is $1/2^n$, similarly as in **Case 2c** .

Case 4d: $A = A'$, $t = t'$ and M, M' differ in blocks with index i, $i < t$. The probability of collision is $1/2^n$, similarly as in **Case 2d** .

Case 4e: $A = A'$, $t = t'$ and M, M' differ only in last block. The probability of collision is 0, similarly as in **Case 2e**.

We deduce that for any two queries (N,A,M) and (N,A',M') we have that

$$\Pr\left[\mathsf{coll}\left(\left(N^h, A^{h,i}, M^{h,i}\right), \left(N^h, A^{h,j}, M^{h,j}\right)\right)\right] \leq \frac{1}{2^n}.$$

Using this result, we can bound the advantage $\mathbf{Adv}^{\mathbf{prf}}_{H^{\tau}_{\widetilde{R}}}(\mathscr{A})$:

$$\mathbf{Adv}^{\mathbf{prf}}_{H^{\tau}_{\widetilde{R}}}(\mathscr{A}) \leq \max\left\{\sum_{h=1}^{r}\sum_{1\leq i<j\leq q_h} \Pr\left[\mathsf{coll}\left(\left(N^h, A^{h,i}, M^{h,i}\right), \left(N^h, A^{h,j}, M^{h,j}\right)\right)\right]\right\}$$
$$\leq \max\left\{\sum_{h=1}^{r}\sum_{1\leq i<j\leq q_h} \frac{1}{2^n}\right\} \leq \max\left\{\sum_{h=1}^{r}\frac{q_h^2}{2}\cdot\frac{1}{2^n}\right\} \leq \frac{0.5\sigma^2}{2^n}$$

with the maximum taken over the choice of $r, q_1, q_2, \dots, q_r$ and the choice of the queries $(N^{1,1}, A^{1,1}, M^{1,1}), \dots, (N^{r,q_r}, A^{r,q_r}, M^{r,q_r})$ such that their total length in blocks is limited by σ, i.e. $\sum_{i=1}^{r}\sum_{j=1}^{q_i}\left(|A^{i,j}|_b + \max\left(1, |M^{i,j}|_b\right)\right) \leq \sigma$. This completes the proof. $\square$

Before we proceed, we have to introduce a new notation. The purpose of this notation is to make the security analysis more comprehensible. Consider the encryption algorithm $\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R},\tau].\mathcal{E}_K(N,A,M)$. The algorithm can be split into two parts. First, it computes $\mathsf{IV} = \mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R},\tau].\mathrm{HASH}_K(N,A,M)$. The second part comprises all the steps after computing the IV. We will denote the second step as $\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R},\tau].\bar{\mathcal{E}}_K(\mathsf{IV},M)$, so that, if we simplify the notation, we have

$$\mathcal{E}_K(N,A,M) = \bar{\mathcal{E}}_K(\mathrm{HASH}_K(N,A,M),M).$$

We define $\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R},\tau].\bar{\mathcal{D}}_K(\mathsf{IV},M)$ in a similar manner.

Lemma 4.3. *Let $\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R},\tau]$ be the MR-OMD scheme that uses tweakable RF $\widetilde{R}$. Let $\mathscr{A}$ be a CPA adversary that makes q_e encryption queries. Then*

$$\mathbf{Adv}^{\mathbf{priv\$}}_{\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R},\tau].\bar{\mathcal{E}}}(\mathscr{A}) \leq \frac{0.5q_e^2}{2^\tau}$$

Proof. For the sake of readability, we will use Π to refer to $\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R},\tau].\bar{\mathcal{E}}$ throughout this proof. We observe the advantage of the adversary $\mathscr{A}$ in two mutually exclusive cases:

$$\mathbf{Adv}^{\mathbf{priv\$}}_{\Pi}(\mathscr{A}) = \mathbf{Adv}^{\mathbf{priv\$|\mathrm{IVcoll}}}_{\Pi}(\mathscr{A})\Pr\left[\mathrm{IVcoll}\right] + \mathbf{Adv}^{\mathbf{priv\$|\neg\mathrm{IVcoll}}}_{\Pi}(\mathscr{A})\Pr\left[\neg\mathrm{IVcoll}\right]$$

where IVcoll denotes the event, that there is a collision among IVs and:

$$\mathbf{Adv}^{\mathbf{priv\$|\mathrm{IVcoll}}}_{\Pi}(\mathscr{A}) = \Pr\left[K \leftarrow_\$ \mathcal{K}:\ \mathscr{A}^{\Pi^\$_K(\cdot)} \Rightarrow 1 \middle| \mathrm{IVcoll}\right] - \Pr\left[\mathscr{A}^{\$(\cdot)} \Rightarrow 1 \middle| \mathrm{IVcoll}\right]$$
$$\mathbf{Adv}^{\mathbf{priv\$|\neg\mathrm{IVcoll}}}_{\Pi}(\mathscr{A}) = \Pr\left[K \leftarrow_\$ \mathcal{K}:\ \mathscr{A}^{\Pi^\$_K(\cdot)} \Rightarrow 1 \middle| \neg\mathrm{IVcoll}\right] - \Pr\left[\mathscr{A}^{\$(\cdot)} \Rightarrow 1 \middle| \neg\mathrm{IVcoll}\right]$$

First, consider the case that there is no collision on IVs. This implies, that all tweaks (IV_j, i) for $1 \leq j \leq q_e$ used to encrypt the queried messages $M^1, \dots, M^{q_e}$ are distinct and thus all the images under the RFs $\widetilde{R}^{\mathsf{IV}_j,i}$, used in the encryption queries, are uniform and independent. Thus, all the ciphertexts $\mathbb{C}^1, \dots, \mathbb{C}^{q_e}$ the adversary $\mathscr{A}$ sees are independent random strings. We deduce $\mathbf{Adv}^{\mathbf{priv\$|\neg\mathrm{IVcoll}}}_{\Pi_K}(\mathscr{A}) = 0$.

We bound $\mathbf{Adv}_{\Pi}^{\mathbf{priv}\$|\mathrm{IVcoll}}(\mathscr{A})\Pr[\mathrm{IVcoll}]$ by the probability $\Pr[\mathrm{IVcoll}]$. We have

$$\Pr[\mathrm{IVcoll}] \leq \sum_{1\leq i<j\leq q_e} \Pr[\mathrm{IV}^{\mathrm{i}} = \mathrm{IV}^{\mathrm{j}}] \leq \sum_{1\leq i<j\leq q_e} \frac{1}{2^\tau} \leq \frac{0.5q_e^2}{2^\tau}$$

We deduce $\mathbf{Adv}_{\Pi}^{\mathbf{priv}\$|\mathrm{IVcoll}}(\mathscr{A}) \leq \frac{0.5q_e^2}{2^\tau}$. This concludes the proof. □

Lemma 4.4. *Let $\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R},\tau]$ be the MR-OMD scheme that uses (tweakable) RF $\widetilde{R} \leftarrow_\$ \widetilde{\mathrm{Func}}(\mathcal{T}, 2^{m+n}, 2^n)$. Let $\mathscr{A}$ be a CCA adversary attacking $\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R},\tau]$. Let q_e be the number of encryption queries and q_d the number of decryption queries made by $\mathscr{A}$ and let σ be the total number of calls to the underlying tweakable RF $\widetilde{R}$ in all $\mathscr{A}$'s queries. Then there exist adversaries $\mathscr{B}$ and $\mathscr{C}$, such that*

$$\mathbf{Adv}^{\mathbf{prf}}_{\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R},\tau].HASH}(\mathscr{C}) + \mathbf{Adv}^{\mathbf{priv}\$}_{\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R},\tau].\bar{\mathcal{E}}}(\mathscr{B}) \geq \mathbf{Adv}^{\mathbf{mrae}}_{\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R},\tau]}(\mathscr{A}) - \frac{q_d}{2^\tau}$$

where $\mathscr{B}$ asks at most q_e queries and $\mathscr{C}$ asks at most $q = q_e + q_d$ queries in total. Both $\mathscr{B}$ and $\mathscr{C}$ are limited to a total number σ of calls to underlying tweakable RF $\widetilde{R}$ in all their queries.

Proof. For the sake of readability, we shall refer to $\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R},\tau]$ by Π throughout this proof. The proof proceeds in two steps, similarly as the security analysis of SIV [RS06a].

In the first step, we start with the scheme $\bar{\Pi}$, which is the same as Π, except that we replace the algorithm Π.HASH by a random function $\rho \leftarrow_\$ \mathrm{Func}(\mathcal{N}\times\mathcal{A}\times\mathcal{M}, \{0,1\}^\tau)$. We still use the (tweakable) RF $\widetilde{R}$ for $\Pi.\bar{\mathcal{E}}$ in $\bar{\Pi}$, so the key space of $\bar{\Pi}$ is

$$\mathrm{Func}(\mathcal{N}\times\mathcal{A}\times\mathcal{M}, \{0,1\}^\tau) \times \widetilde{\mathrm{Func}}(\mathcal{T}, 2^{m+n}, 2^n)$$

and the key is formed by $\widetilde{R},\rho$. Let $\bar{p} = \mathbf{Adv}_{\bar{\Pi}}^{\mathbf{mrae}}(\mathscr{A})$ with unchanged limits on resources q_e, q_d, σ. We have

$$\begin{aligned}\bar{p} &= \Pr\left[\mathscr{A}^{\mathbf{mrae\text{-}R}_{\bar{\Pi}}} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathbf{mrae\text{-}I}_{\bar{\Pi}}} \Rightarrow 1\right]\\ &= \bar{p}_1 + \bar{p}_2\end{aligned}$$

with

$$\begin{aligned}\bar{p}_1 &= \Pr\left[\mathscr{A}^{\mathbf{mrae\text{-}R}_{\bar{\Pi}}} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathbf{mrae\text{-}H}_{\bar{\Pi}}} \Rightarrow 1\right]\\ \bar{p}_2 &= \Pr\left[\mathscr{A}^{\mathbf{mrae\text{-}H}_{\bar{\Pi}}} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathbf{mrae\text{-}I}_{\bar{\Pi}}} \Rightarrow 1\right]\end{aligned}$$

where the game $\mathbf{mrae\text{-}H}_{\bar{\Pi}}$ is the same as $\mathbf{mrae\text{-}R}_{\bar{\Pi}}$, except the decryption oracle is taken from $\mathbf{mrae\text{-}I}_{\bar{\Pi}}$, i.e. the decryption queries are always replied with $\perp$.

We proceed by bounding the terms $\bar{p}_1$ and $\bar{p}_2$. To bound $\bar{p}_2$, we construct an adversary $\mathscr{B}$ for attacking the priv\$ security of $\bar{\Pi}.\bar{\mathcal{E}}$ from $\mathscr{A}$. $\mathscr{B}$ is equipped with its own oracle $e(\cdot)$, which implements either $\bar{\Pi}.\bar{\mathcal{E}}$ or the random bits oracle. We let $\mathscr{B}$ run $\mathscr{A}$. On

$\mathscr{A}$'s query (N, A, M) to the encryption oracle, $\mathscr{B}$ queries it's own oracle $e(\cdot)$ with M and returns the result to $\mathscr{A}$. On any query from $\mathscr{A}$ to decryption oracle, $\mathscr{B}$ returns $\perp$. When $\mathscr{A}$ halts and outputs bit b, $\mathscr{B}$ stops and outputs b as well.

If $e(\cdot) = \bar{\Pi}.\bar{\mathcal{E}}^{\$}_{\widetilde{R}}(\cdot)$, then $\mathscr{B}$ simulates the game **mrae-H**$_{\bar{\Pi}}$ correctly (the assumption, that $\mathscr{A}$ does not repeat queries, is needed here). If $e(\cdot) = \$(\cdot)$, then $\mathscr{B}$ correctly simulates the game **mrae-I**$_{\bar{\Pi}}$. We deduce $\bar{p}_2 \leq \mathbf{Adv}^{\mathbf{priv\$}}_{\Pi.\bar{\mathcal{E}}}(\mathscr{B})$.

To give a bound on $\bar{p}_1$, we shall reveal the tweakable RF $\widetilde{R}$ to $\mathscr{A}$. Clearly, an upper bound of the advantage in this case will also be valid if $\mathscr{A}$ does not have $\widetilde{R}$, since having $\widetilde{R}$ only makes the attack easier:

$$\begin{aligned}\bar{p}_1 =& \Pr\left[\mathscr{A}^{\textbf{mrae-R}_{\bar{\Pi}}} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\textbf{mrae-H}_{\bar{\Pi}}} \Rightarrow 1\right] \\ \leq& \Pr\left[\mathscr{A}(\widetilde{R})^{\textbf{mrae-R}_{\bar{\Pi}}} \Rightarrow 1\right] - \Pr\left[\mathscr{A}(\widetilde{R})^{\textbf{mrae-H}_{\bar{\Pi}}} \Rightarrow 1\right]\end{aligned}$$

In this setting, $\mathscr{A}$ can only tell the difference between the two games, if the decryption query returns something other than $\perp$ (then $\mathscr{A}$ stops and outputs 1). This happens, if $\mathscr{A}$ builds a query $(A, \mathbb{C})$ to $\bar{\Pi}^{-1}_{\widetilde{R},\rho}$, that successfully verifies and decrypts, and that happens if $\mathsf{IV} = \rho(N, A, M)$ and $M = \bar{\Pi}.\bar{\mathcal{D}}_{\widetilde{R}}(\mathsf{IV}, \mathbb{C})$. Recall that the adversary is assumed not to query its decryption oracle with $(N, A, \mathbb{C})$ if it had previously obtained $\mathbb{C}$ from an encryption query (N, A, M). Having $\widetilde{R}$, $\mathscr{A}$ can compute $M = \bar{\Pi}.\bar{\mathcal{D}}_{\widetilde{R}}(\mathsf{IV}, \mathbb{C})$ for any pair $\mathsf{IV}, \mathbb{C}$, but it never knows a pair $\mathsf{IV}, (N, A, M)$, s.t. $\mathsf{IV} = \rho(N, A, M)$ and (N, A, M) has not been queried to the encryption oracle before. $\mathscr{A}$ is thus left to guess the correct IV and the probability of producing a decryption query, that does not result in $\perp$ is at most $1/2^{\tau}$. If we consider all queries made by $\mathscr{A}$, we have $\bar{p}_1 \leq q_d/2^n$. We then have $\bar{p} \leq \mathbf{Adv}^{\mathbf{priv\$}}_{\Pi.\bar{\mathcal{E}}}(\mathscr{B}) + q_d/2^{\tau}$.

At the beginning of the second step of the proof, we point out that

$$\mathbf{Adv}^{\mathbf{mrae}}_{\Pi}(\mathscr{A}) = \Pr\left[\mathscr{A}^{\textbf{mrae-R}_{\Pi}} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\textbf{mrae-R}_{\bar{\Pi}}} \Rightarrow 1\right] + \bar{p}$$

as the games **mrae-I**$_{\Pi}$ and **mrae-I**$_{\bar{\Pi}}$ are identical. We construct an adversary $\mathscr{C}$ that attacks Π.HASH as a PRF (keyed by a (tweakable) RF $\widetilde{R}$), such that it uses $\mathscr{A}$ as a subroutine. $\mathscr{C}$ is equipped with its own oracle $r(\cdot,\cdot,\cdot)$, which implements either Π.HASH or a corresponding RF ρ. The adversary $\mathscr{C}$ picks $\widetilde{R} \leftarrow\$ \widetilde{\mathrm{Func}}(\mathcal{T}, 2^{m+n}, 2^n)$ and runs $\mathscr{A}$. On $\mathscr{A}$'s encryption query (N, A, M), $\mathscr{C}$ sets $\mathsf{IV} \leftarrow r(N, A, M)$, computes $C \leftarrow \Pi.\bar{\mathcal{E}}_{\widetilde{R}}(IV, M)$ and returns $\mathsf{IV}||C$ to $\mathscr{A}$. When $\mathscr{A}$ asks a decryption query $(N, A, \mathsf{IV}||C)$, $\mathscr{C}$ first computes $M \leftarrow \Pi.\bar{\mathcal{D}}_{\widetilde{R}}(\mathsf{IV}, C)$, then it returns M to $\mathscr{A}$ only if $\mathsf{IV} = r(N, A, M)$, otherwise it returns $\perp$. When $\mathscr{A}$ stops and outputs bit b, let $\mathscr{B}$ stop and output b.

It is easy to see, that if $r = \rho$, then $\mathscr{C}$ correctly simulates **mrae-R**$_{\bar{\Pi}}$. It remains to show, that if $r = \Pi.\mathrm{HASH}_{\widetilde{R'}}$ is the construction Π.HASH keyed by an independent $\widetilde{R'} \in \widetilde{\mathrm{Func}}(\mathcal{T}, 2^{m+n}, 2^n)$, the adversary $\mathscr{C}$ simulates the oracles $\Pi_{\widetilde{R^*}}(\cdot,\cdot,\cdot), \Pi^{-1}_{\widetilde{R^*}}(\cdot,\cdot,\cdot)$ in the game **mrae-R**$_{\Pi}$ correctly for some $\widetilde{R^*} \in \mathrm{Func}^{\mathcal{T}}(m+n, n)$.

If we indeed have that $r = \Pi.\mathrm{HASH}_{\widetilde{R'}}$, then the game for $\mathscr{C}$ has picked the RF

$\widetilde{R'} \leftarrow\!\$\ \widetilde{\mathrm{Func}}(\mathcal{T}, 2^{m+n}, 2^n)$, while $\mathscr{C}$ has picked the tweakable RF $\widetilde{R} \leftarrow\!\$\ \widetilde{\mathrm{Func}}(\mathcal{T}, 2^{m+n}, 2^n)$ independently. The construction of Π is such, that the set of tweaks $\mathcal{T}_e = \mathcal{IV} \times \mathbb{N}$ used in $\Pi.\bar{\mathcal{E}}$ is disjoint with the set of tweaks $\mathcal{T}_h = \mathcal{T} \setminus \mathcal{T}_e$ used in Π.HASH. For every $R, R' \in \mathrm{Func}^{\mathcal{T}}(m+n, n)$ there is some $R^* \in \mathrm{Func}^{\mathcal{T}}(m+n, n)$ such that

$$R^{*\mathsf{T}_e}(\cdot) = R^{\mathsf{T}_e}(\cdot) \text{ for all } \mathsf{T}_e \in \mathcal{T}_e, \quad \text{and} \quad R^{*\mathsf{T}_h}(\cdot) = R'^{\mathsf{T}_h}(\cdot) \mathrm{forall} \mathsf{T}_h \in \mathcal{T}_h,$$

so the oracles simulated by $\mathscr{C}$ are equivalent with oracles $\Pi_{\widetilde{R^*}}(\cdot,\cdot,\cdot), \Pi^{-1}_{\widetilde{R^*}}(\cdot,\cdot,\cdot)$. We let $R, R' \leftrightarrow R^*$ denote that three (tweakable) functions $R, R', R^* \in \widetilde{\mathrm{Func}}(\mathcal{T}, 2^{m+n}, 2^n)$ have the property just described. It remains to show, that the distribution of $\widetilde{R^*}$ is uniform assuming that $\widetilde{R}, \widetilde{R'}$ are distributed uniformly and independently in $\widetilde{\mathrm{Func}}(\mathcal{T}, 2^{m+n}, 2^n)$. We have

$$\begin{aligned}
\Pr\left[\widetilde{R^*} = R^*\right] &= \sum_{R,R' \colon R,R' \leftrightarrow R^*} \Pr\left[\widetilde{R} = R, \widetilde{R'} = R'\right] \\
&= \sum_{R,R' \colon R,R' \leftrightarrow R^*} \left(\frac{1}{2^{n\cdot|\mathcal{T}|\cdot 2^{m+n}}}\right)^2 \\
&= \left|\left\{R, R' | R, R' \leftrightarrow R^*\right\}\right| \cdot \left(\frac{1}{2^{n\cdot|\mathcal{T}|\cdot 2^{m+n}}}\right)^2 \\
&= 2^{n\cdot|\mathcal{T}_h|\cdot 2^{m+n}} \cdot 2^{n\cdot|\mathcal{T}_e|\cdot 2^{m+n}} \cdot \left(\frac{1}{2^{n\cdot|\mathcal{T}|\cdot 2^{m+n}}}\right)^2 \\
&= \frac{1}{2^{n\cdot|\mathcal{T}|\cdot 2^{m+n}}}
\end{aligned}$$

so the distribution of $\widetilde{R^*}$ is indeed uniform, and simulation of $\mathscr{A}$'s oracles is correct. We deduce $\mathbf{Adv}^{\mathbf{mrae}}_{\Pi}(\mathscr{A}) \le \bar{p} + \mathbf{Adv}^{\mathbf{prf}}_{\Pi.\mathrm{HASH}}(\mathscr{C})$. This concludes the proof. □

Instantiating a (tweakable) RF with a PRF. The proof of Theorem 4.1 is completed in the same way as with OMD (Section 3.6). First, the (tweakable) RF $\widetilde{R}$ is replaced by a (tweakable) PRF $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$, where $\mathcal{K} = \{0,1\}^k$. This will increase the security bound as shown in Lemma 4.5.

Lemma 4.5. *Let $\widetilde{R} \leftarrow\!\$\ \widetilde{\mathrm{Func}}(\mathcal{T}, 2^{m+n}, 2^n)$ be a (tweakable) RF and $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be a (tweakable) PRF. Let $\mathscr{A}$ be a CCA adversary that runs in time t, makes q_e encryption queries and q_d decryption queries that induce no more than σ calls to $\widetilde{F}$ in total. Then*

$$\mathbf{Adv}^{\mathbf{mrae}}_{\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{F},\tau]}(\mathscr{A}) \le \mathbf{Adv}^{\mathbf{mrae}}_{\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R},\tau]}(\mathscr{B}) + \mathbf{Adv}^{\widetilde{\mathbf{prf}}}_{\widetilde{F}}(\mathscr{C})$$

for some information theoretic $\mathscr{B}$ that makes q_e encryption queries and q_d decryption queries that induce no more than σ calls to $\widetilde{R}$ in total, and some $\mathscr{C}$ that makes no more than σ queries and runs in time $t + \gamma \cdot n \cdot \sigma$ for some constant γ.

Proof. The proof of this lemma is very similar to the proof of the PRIV bound in Lemma 3.3, we omit the details. □

We instantiate the (tweakable) PRF $\widetilde{F}$ from a PRF F (with a smaller domain) by the means of xoring a mask to a part of the input of F, exactly as with OMD. The tweaks in MR-OMD are either of the form $\mathsf{T} = (\alpha, i, j)$ where $\alpha \in \mathcal{N}$, $1 \le i \le 2^{n-5}$ and $j \in \{0, \ldots, 5\}$ or of the form $\mathsf{T}' = (\mathsf{IV}, i)$ with $\alpha \in \mathcal{IV}$, $1 \le i \le 2^{n-5}$. We can have a unified notation for all the tweaks as $\mathsf{T} = (\alpha, i, j)$ where $\alpha \in \mathcal{N} \cup \mathcal{IV}$, $1 \le i \le 2^{n-5}$ and $j \in \{0, \ldots, 5\}$ if $\alpha \in \mathcal{N}$ and $j = 6$ if $\alpha \in \mathcal{IV}$.

The transition from tweakable PRFs to PRFs with xor-masks being exactly the same, we heavily rely on the security analysis from Section 3.6.

Lemma 4.6. *Let $\widetilde{F} : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be a function family with key space $\mathcal{K}$. Let $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be defined by $\widetilde{F}_K^{\mathsf{T}}(X \| Y) = F_K((X \oplus \Delta(\mathsf{T})) \| Y)$ for every $\mathsf{T} \in \mathcal{T}, K \in \mathcal{K}$, $X \in \{0,1\}^n, Y \in \{0,1\}^m$ and let $\Delta_K(\mathsf{T})$ be the masking function of MR-OMD as defined in Section 4.4. Let $\mathscr{A}$ be an adversary that runs in time t and makes q queries. Then*

$$\mathbf{Adv}_{\widetilde{F}}^{\mathbf{prf}}(\mathscr{A}) \le \mathbf{Adv}_{F}^{\mathbf{prf}}(\mathscr{B}) + \frac{3q^2}{2^n}$$

for some $\mathscr{B}$ that runs in time $t + \gamma \cdot n \cdot q$ and makes $2 \cdot q$ queries.

Proof. The proof of this lemma is almost identical to the proof of Lemma 3.4; we just need to show that the masking function of MR-OMD $\Delta_K(\mathsf{T}) = \Delta_K(\alpha, i, j)$ is also a 2^{-n}-uniform 2^{-n}-AXU hash.

It is easy to verify that it is the case. Compared to the masking function of OMD, the j-component of the tweak input can now take values up to 6, and its binary representation will thus have at most three non-zero least-significant bits. This is taken care of by the fact that $L[0] = 2^3 \cdot L_*$ (instead of $2^2 \cdot L_*$ as in OMD). Also, we impose $i \le 2^{n-5}$ (instead of $i \le 2^{n-4}$ as in OMD), so for every unique (i, j), $(2^3 \cdot \gamma_i \oplus \langle j \rangle_n)$ is a unique element of $\mathsf{GF}(2^n)$. Here γ_i is the i^{th} codeword of the canonical Gray code defined in the proof of Lemma 3.4.

With this property, the rest of the analysis of Lemma 3.4 carries over and yields the desired result. □

Security in the Nonce-Respecting Case. Intuitively, one would expect that the security bound in the nonce-respecting setting should be somewhat better than the one in the nonce-reuse case. Theorem 4.7 gives a bound on the AE security of MR-OMD in the nonce-respecting scenario, confirming this intuition.

Theorem 4.7. *Fix $n \ge 1$ and $\tau \in \{0, 1, \cdots, n\}$. Let $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be a PRF, where $1 \le m \le n$. Let $\mathscr{A}$ be a CCA adversary that runs in time t, and makes q_e encryption queries and q_d decryption queries that induce no more than σ calls to F*

in total. Then

$$\mathbf{Adv}^{\mathbf{nae}}_{\text{MR-OMD}[F,\tau]}(\mathscr{A}) \le \mathbf{Adv}^{\mathbf{prf}}_{F}(\mathscr{B}) + \frac{3\sigma^2}{2^n} + \frac{0.5q_e^2}{2^\tau} + \frac{q_d}{2^\tau}$$

for some $\mathscr{B}$ that makes no more than $2 \cdot q$ queries and runs in time $t + \gamma \cdot n \cdot \sigma$ for some constant γ.

Proof. The steps to prove this theorem are almost the same as for Theorem 4.1. The only difference is in the proof for the security of the HASH algorithm as a PRF. This is easy to see, as HASH is the only component of MR-OMD where the nonce is used. Lemma 4.8 gives the PRF security bound for HASH in the nonce-respecting setting. The bound stated in Theorem 4.7 is obtained combining Lemma 4.8 with Lemma 4.4, Lemma 4.3, Lemma 4.5 and Lemma 4.6. □

Lemma 4.8. *Let $\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R}, \tau]$ be the MR-OMD scheme that uses (tweakable) RF $\widetilde{R}$. Let $\mathscr{A}$ be a nonce-respecting adversary that makes at most q queries that induce no more than σ calls to $\widetilde{R}$ in total. Then*

$$\mathbf{Adv}^{\mathbf{prf}}_{\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R},\tau].HASH}(\mathscr{A}) = 0.$$

Proof. Recall the proof of Lemma 4.2: the $\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R}, \tau]$.HASH behaves as a RF unless there is a collision in the input to the final RF. This is because the final random function, determined by the final tweak, may be the same for several messages.

Now, considering that adversaries are nonce-respecting, we have that for every query $(N^i, A^i, M^i), 1 \le i \le q$ made by the adversary the nonce is distinct, i.e. $N^i \ne N^j$ if $i \ne j$. Each query is processed using a subset of $\mathcal{T}$, that is disjoint with tweak sets used to process all the other queries. Therefore, when a query is processed, the final tweak is always fresh (never used before) and the random function is independent from all others so far. The distribution produced by $\mathbb{MR}\text{-}\mathbb{OMD}[\widetilde{R}, \tau]$.HASH is then identical with that produced by a random function $\rho \leftarrow\$ \text{Func}(\mathcal{N} \times \mathcal{A} \times \mathcal{M}, \{0,1\}^\tau)$. □

4.6 Parallelizable MR-OMD

The MR-OMD scheme described in section 4.4 is designed to be substantially similar to OMD, so that it is able to share a lot of common software/hardware with OMD, while achieving stronger security goals than OMD itself. This similarity also implies that the encryption (and decryption) in MR-OMD is kept serial as it is in OMD. However, we notice that the two-pass construction (in contrast to OMD which is one-pass) also opens up the possibility of having a parallelizable version of the encryption (and decryption) algorithm.

The IV in MR-OMD is computed from both associated data and message using a PRF. Thus, the encryption is always dependent on the whole query (via IV) and we no longer need to apply the serial, chaining encryption of OMD. So, in specific applications where there are possibilities for parallel computation, we might want to modify MR-OMD to exploit this fact. For this purpose, we propose PMR-OMD. PMR-OMD uses the same algorithms Initialize and HASH as MR-OMD, but the encryption (and the decryption) algorithm uses counter mode. Schematic visualisation can be found in Figure 4.1. This replacement will of course get us further from the original OMD, which may be inconvenient in hardware implementations; however, in software implementations, the parallel execution might be exactly what we want, especially for general purpose CPUs with multiple cores. The PMR-OMD is almost fully parallelizable, with a single bottleneck in the form of the call to the compression function when processing the final message block in the HASH algorithm.

Security of PMR-OMD. The security bounds of PMR-OMD are exactly the same as those of MR-OMD, both in the nonce-misusing and nonce-respecting settings. This is because the proof of \$-privacy of the counter mode is essentially the same as the one of \$-privacy of the original OMD encryption. The remaining components of MR-OMD (and thus also the proofs) remain unchanged. We therefore omit the proofs of the following theorems.

Theorem 4.9. *Fix $n \geq 1$ and $\tau \in \{0, 1, \cdots, n\}$. Let $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be a keyed function, where $1 \leq m \leq n$. Let $\mathscr{A}$ be a CCA adversary that runs in time t, makes q_e encryption queries and q_d decryption queries that induce no more than σ calls to F in total. Then*

$$\mathbf{Adv}^{\mathbf{mrae}}_{PMR\text{-}OMD[F,\tau]}(\mathscr{A}) \leq \mathbf{Adv}^{\mathbf{prf}}_{F}(\mathscr{B}) + \frac{3.5\sigma^2}{2^n} + \frac{0.5q_e^2}{2^\tau} + \frac{q_d}{2^\tau}$$

for some $\mathscr{B}$ that runs in time $t + \gamma \cdot n \cdot \sigma$ for some constant γ and makes $2 \cdot \sigma$ queries.

Theorem 4.10. *Fix $n \geq 1$ and $\tau \in \{0, 1, \cdots, n\}$. Let $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be a PRF, where $1 \leq m \leq n$. Let $\mathscr{A}$ be a CCA adversary that runs in time t, makes q_e encryption queries and q_d decryption queries that induce no more than σ calls to F in total. Then*

$$\mathbf{Adv}^{\mathbf{nae}}_{PMR\text{-}OMD[F,\tau]}(\mathscr{A}) \leq \mathbf{Adv}^{\mathbf{prf}}_{F}(\mathscr{B}) + \frac{3\sigma^2}{2^n} + \frac{0.5q_e^2}{2^\tau} + \frac{q_d}{2^\tau}$$

for some $\mathscr{B}$ that makes no more than $2 \cdot q$ queries and runs in time $t + \gamma \cdot n \cdot \sigma$ for some constant γ.

Chapter 5

Boosting OMD for Almost Free Authentication of Associated Data

This chapter is dedicated to pure OMD (p-OMD) which is a variant of OMD that has improved performance.

The work presented in this chapter is a result of joint work with Reza Reyhanitabar and Serge Vaudenay which was published in FSE 2015 [RVV15].

Organization of the Chapter. We briefly discuss the related work in Section 5.1 and give a summary of the contribution in Section 5.2.

We introduce pOMD in Section 5.3, give its description in Section 5.4, give its security analysis in Section 5.5, and discuss its performance in Section 5.6.

5.1 Related Work

The security notion targeted by p-OMD is the same as for OMD, NAE security [Rog02]. The integration of AD-blocks into the message processing core of OMD is inspired by the BNMAC by Yasuda [Yas07]. The first version of the publication that introduced p-OMD claimed that p-OMD preserved authenticity under nonce misuse. This claim was refuted by Ashur and Mennink who then proposed two simplified variants of p-OMD, one of which did achieve the desired misuse resistant property [AM16].

5.2 Contribution

We design p-OMD, a variant of OMD that integrates the AD processing into the core encryption algorithm of OMD. The algorithmic modifications that transforms OMD to p-OMD make the AD processing almost free in terms of spent computational time, and we prove that they have no adverse impact on security.

We theoretically predict and then experimentally verify the speedup of p-OMD compared to OMD.

5.3 Pure OMD

The original OMD scheme couples a single pass of the modified Merkle-Damgård (MD) iteration (in which the intermediate chaining values are xored with specially crafted offsets) with the counter-based XOR MAC algorithm [BGR95] to process a message and its associated data. In this chapter, we investigate the possibility of making algorithmic improvements to the original OMD scheme, aiming at boosting its efficiency, while preserving its security properties.

In particular, we provide a positive answer to the question “can we dispense with the XOR MAC algorithm, and make the scheme based purely on Merkle-Damgård?” We show that there is a natural way (inspired by the work by Yasuda [Yas07]) to modify OMD to make it more compact and efficient with respect to processing associated data. Our new variant of OMD, called *pure* OMD (p-OMD), has the following features:

It inherits all desirable security features of OMD. We prove the security of p-OMD under the same standard assumption (namely, pseudo-randomness of the compression function) as used for OMD. Furthermore, the proven security bounds for p-OMD are the same as those of OMD. This shows that the modifications we made to OMD, to obtain the performance-boosted variant p-OMD, are without sacrificing any security.

It has a more compact structure and processing AD is almost free. The p-OMD scheme dispenses with the XOR MAC algorithm and is solely based on the (masked) MD iteration. This is achieved by absorbing the associated data blocks during the core MD path rather than processing them separately by an additional algorithm. To encrypt a message of ℓ blocks together with associated data of a blocks, OMD needs $\ell+a+2$ calls to the compression function while p-OMD only requires $\max\{\ell, a\}+2$ calls. That is, for a typical case where $\ell \geq a$, p-OMD makes just $\ell + 2$ calls independently of the length of AD.

5.4 Description

The AE scheme p-OMD is a mode of operation that converts a keyed compression function to an AEAD scheme. To instantiate p-OMD, one must first choose and fix a keyed compression function $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ and a tag length $\tau \leq n$, such that the key space $\mathcal{K} = \{0,1\}^k$ for an integer k and $m \leq n$. Let p-OMD$[F, \tau]$ denote p-OMD instantiated by fixing F and τ.

An instance p-OMD$[F, \tau]$ has a key space $\mathcal{K} = \{0,1\}^k$, an AD space $\mathcal{A} \subseteq \{0,1\}^{\leq (n) \cdot 2^{n-6}}$ and a message space $\mathcal{M} \subseteq \{0,1\}^{\leq m \cdot 2^{n-6}}$. We let $\ell_{max} = \max_{X \in \mathcal{A} \cup \mathcal{M}}(|X|_m)$ denote the upper bound on the maximum number of blocks in any single message or AD. We require that the nonce space $\mathcal{N} = \{0,1\}^\nu$ for an integer $1 \le \nu < n$.

An overview. The main design rationale behind p-OMD is the integration of AD processing into the same MD path that processes the message. While the overall structure of such design is rather simple, the combined processing of the message and associated data blocks in p-OMD creates a not-so-small number of cases that occur during the encryption due to the possible lengths (and relative lengths) of messages and AD, and need to be treated and analyzed carefully.

Figure 5.1 shows an illustration of the encryption algorithm of p-OMD$[F, \tau]$. The decryption algorithm can be straightforwardly derived from the encryption algorithm with the additional verification of the authentication tag at the end of the decryption process. Figures 5.2 and 5.3 provide a complete algorithmic description ofthe encryption algorithm of pOMD.

In the following, we briefly explain the main components of pOMD in more detail.

Computing $\boldsymbol{\Delta_{N,i,j}}$. As shown in Figure 5.1, before each call to the underlying compression function F, we xor a (key-dependent) masking value $\Delta_{N,i,j}$ to a part fo the input, where N is the nonce, the i component is incremented at each call to the compression function and the j component is changed when needed (according to a pattern that will be detailed shortly). In the following, all multiplications are in $GF(2^n)$.

Precomputation. Unlike in OMD, we precompute an array of L_* values. We let $L_*[0] = 0^n$, define $L_*[1] = F_K(0^n, 0^m)$ and let $L_*[i] = i \cdot L_*[1]$ for $2 \le i \le 15$.

We let $L[0] = 2^4 \cdot L_*[1]$ and $L[j] = 2 \cdot L[j-1]$ for $j \ge 1$. For a fast implementation, the values $L_*[i]$ and $L[j]$ can be precomputed and stored in a table for $1 \le i \le 15$ and $0 \le j \le \lceil \log_2(\ell_{max}) \rceil$. Alternatively, (if there is a memory restriction) they can be computed on-the-fly.

Note that all $L_*[i]$ are linear functions of $L_*[1]$. Thus $L_*[\mathsf{int}(\langle j \rangle_4 \oplus \langle j' \rangle_4)] = L_*[j] \oplus L_*[j']$ for any $j, j' \in \{0, 1, \ldots, 15\}$ (with a slight abuse of notation).

Computation of the masking sequence. The masking values $\Delta_{N,i,j}$ are computed sequentially as follows. We define $\Delta_{N,0,0} = F_K(N \| 10^{n-1-\nu}, 0^m)$. To increment i, we let

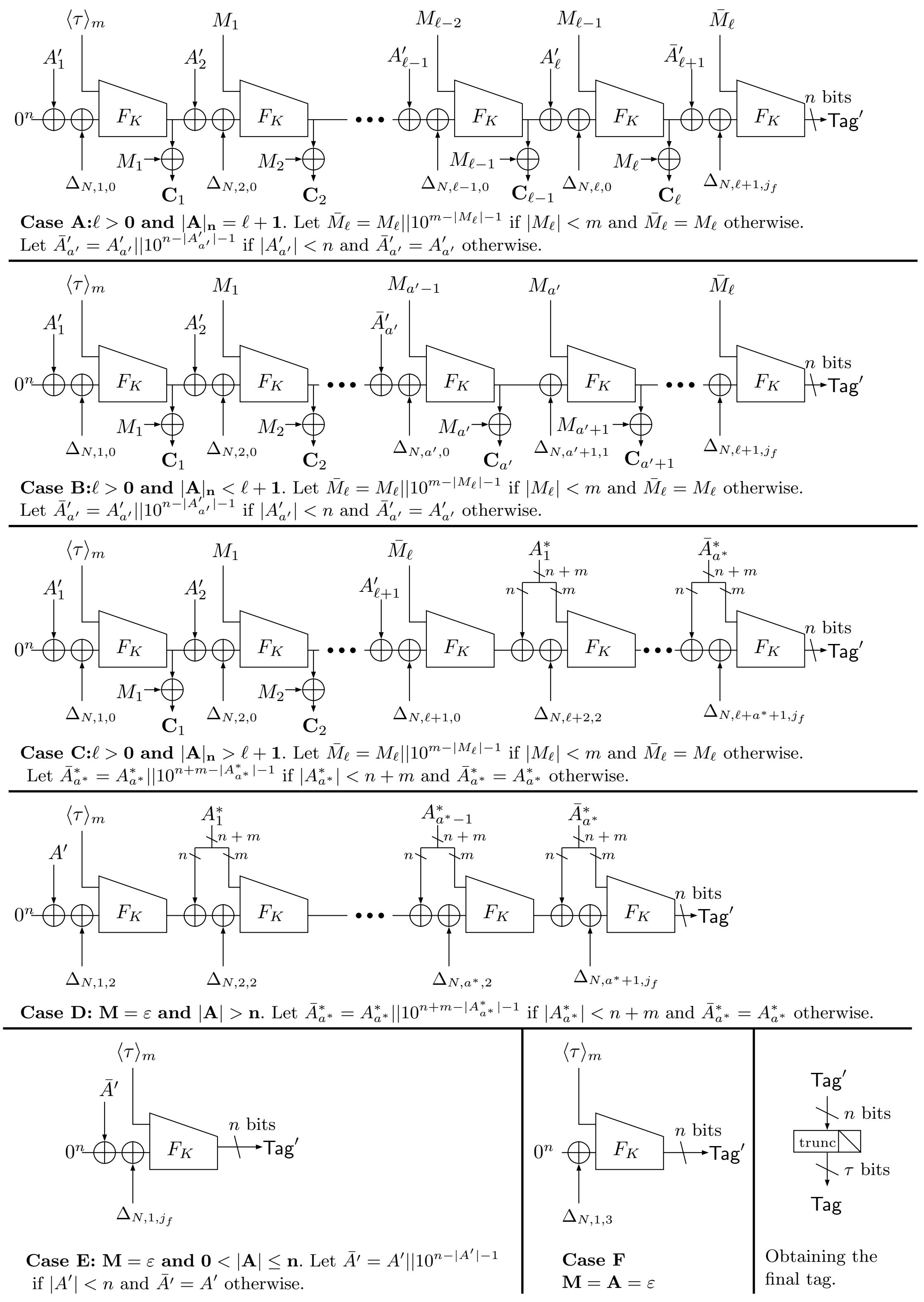
$\Delta_{N,i,j} = \Delta_{N,i-1,j} \oplus L[\mathtt{ntz}(i)]$

for $i \ge 1$ and any $j \in \{0, 1, \ldots, 15\}$. To switch j, we let

$\Delta_{N,i,j} = \Delta_{N,i,j'} \oplus L_*[\mathsf{int}(\langle j \rangle_4 \oplus \langle j' \rangle_4)]$

for any $j, j' \in \{0, 1, \ldots, 15\}$. For details on how we get this compact relation based on the Gray code sequence, we refer to Appendix A.2.

Encryption algorithm. To encrypt a message $M \in \mathcal{M}$ with associated data $A \in \mathcal{A}$ using nonce $N \in \mathcal{N}$ and key $K \in \mathcal{K}$, obtaining a ciphertext $\mathbb{C} = C \| T \in \{0,1\}^{|M|+\tau}$, we carry out the following steps.



Case A:$\ell > 0$ **and** $|\mathbf{A}|_\mathbf{n} = \ell + \mathbf{1}$. Let $\bar{M}_\ell = M_\ell||10^{m-|M_\ell|-1}$ if $|M_\ell| < m$ and $\bar{M}_\ell = M_\ell$ otherwise. Let $\bar{A}'_{a'} = A'_{a'}||10^{n-|A'_{a'}|-1}$ if $|A'_{a'}| < n$ and $\bar{A}'_{a'} = A'_{a'}$ otherwise.

Case B:$\ell > 0$ **and** $|\mathbf{A}|_\mathbf{n} < \ell + \mathbf{1}$. Let $\bar{M}_\ell = M_\ell||10^{m-|M_\ell|-1}$ if $|M_\ell| < m$ and $\bar{M}_\ell = M_\ell$ otherwise. Let $\bar{A}'_{a'} = A'_{a'}||10^{n-|A'_{a'}|-1}$ if $|A'_{a'}| < n$ and $\bar{A}'_{a'} = A'_{a'}$ otherwise.

Case C:$\ell > 0$ **and** $|\mathbf{A}|_\mathbf{n} > \ell + \mathbf{1}$. Let $\bar{M}_\ell = M_\ell||10^{m-|M_\ell|-1}$ if $|M_\ell| < m$ and $\bar{M}_\ell = M_\ell$ otherwise. Let $\bar{A}^*_{a^*} = A^*_{a^*}||10^{n+m-|A^*_{a^*}|-1}$ if $|A^*_{a^*}| < n + m$ and $\bar{A}^*_{a^*} = A^*_{a^*}$ otherwise.

Case D: $\mathbf{M} = \varepsilon$ **and** $|\mathbf{A}| > \mathbf{n}$. Let $\bar{A}^*_{a^*} = A^*_{a^*}||10^{n+m-|A^*_{a^*}|-1}$ if $|A^*_{a^*}| < n + m$ and $\bar{A}^*_{a^*} = A^*_{a^*}$ otherwise.

Case E: $\mathbf{M} = \varepsilon$ **and** $\mathbf{0} < |\mathbf{A}| \leq \mathbf{n}$. Let $\bar{A}' = A'||10^{n-|A'|-1}$ if $|A'| < n$ and $\bar{A}' = A'$ otherwise.

Case F $\mathbf{M} = \mathbf{A} = \varepsilon$

Obtaining the final tag.

Figure 5.1 – **The encryption algorithm of p-OMD$[\boldsymbol{F}, \boldsymbol{\tau}]$.** For the details on how the parameters and masking offsets are computed consult the description in Section 5.4.

```
algorithm Precompute($K$)
    $L_*[0] = 0^n$
    $L_*[1] \leftarrow F_K(0^n, 0^m)$
    for $i \leftarrow 2$ to 15 do
        $L_*[i] = i \cdot L_*[1]$
    end for
    $L[0] \leftarrow 2^4 \cdot L_*[1]$
    for $i \leftarrow 1$ to $\lceil \log_2(\ell_{max}) \rceil$ do
        $L[i] = 2 \cdot L[i-1]$
    end for
    return
end algorithm
```

```
algorithm $\mathcal{E}_K(N, A, M)$
    if $\nu > n-1$ then
        return $\perp$
    end if
    PARTITION($A, M$)
    PAD($A', A^*, M$)
    $\Delta \leftarrow F_K(N \| 10^{n-1-\nu}, 0^m)$
    $\Delta \leftarrow \Delta \oplus L[0]$                          ▷ $\Delta_{N,1,0}$
    $H \leftarrow 0^n$; $j \leftarrow 0$
    if $a' = 0$ and $\ell = 0$ then
        SWITCH($\Delta, j, 3$)
    else if $a' = 0$ then
        SWITCH($\Delta, j, 1$)
    else
        $H \leftarrow H \oplus A'_1$
        if $a' = 1$ and $a^* > 0$ then
            SWITCH($\Delta, j, 2$)
        else if $a' = 1$ and $\ell = 0$ then
            SWITCH($\Delta, j, 12 + j_A + j_M$)
        end if
    end if
    $H \leftarrow F_K(H \oplus \Delta, \langle \tau \rangle_m)$
    $i \leftarrow 2$
    if $a' > 1$ then                                  ▷ STAGE 1
        PROC1($M, A', H, \Delta, i$)
        $C_{i-1} \leftarrow H \oplus M_{i-1}$
        $H \leftarrow H \oplus A'_i$
        $\Delta \leftarrow \Delta \oplus L[\mathtt{ntz}(i)]$
        if $a^* = 0$ and $i = \ell + 1$ then
            SWITCH($\Delta, j, 4 + j_A + j_M$)
            $H \leftarrow F_K(H \oplus \Delta, \bar{M}_{i-1})$
        else
            $H \leftarrow F_K(H \oplus \Delta, M_{i-1})$
        end if
        $i \leftarrow a' + 1$
    end if
    if $\ell \geq a'$ then                               ▷ STAGE 2
        SWITCH($\Delta, j, 1$)
        PROC2($M, H, \Delta, i$)
        $C_{i-1} \leftarrow H \oplus M_{i-1}$
        $\Delta \leftarrow \Delta \oplus L[\mathtt{ntz}(i)]$
        SWITCH($\Delta, j, 8 + j_A + j_M$)
        $H \leftarrow F_K(H \oplus \Delta, \bar{M}_{i-1})$
    else if $a^* > 0$ then                            ▷ STAGE 3
        SWITCH($\Delta, j, 2$)
        PROC3($A^*, H, \Delta, i$)
        $H \leftarrow H \oplus \mathsf{left}_n(A^*_{a^*})$
        $\Delta \leftarrow \Delta \oplus L[\mathtt{ntz}(i)]$
        SWITCH($\Delta, j, 12 + j_A + j_M$)
        $H \leftarrow F_K(H \oplus \Delta, \mathsf{right}_m(A^*_{a^*}))$
    end if
    $T \leftarrow H[n-1 \cdots n-\tau]$
    $\mathbb{C} \leftarrow C_1 \| C_2 \| \cdots \| C_\ell \| T$
    return $\mathbb{C}$
end algorithm
```

Figure 5.2 – **Description of the encryption algorithm of p-OMD[F, τ].** STAGE 1 processes blocks of message and AD simultaneously (**Cases A,B** and **C** in Figure 5.1). STAGE 2 processes only message blocks (**Case B** in Figure 5.1 and the case when we only have a message and no AD that is not illustrated). STAGE 3 processes only double blocks of AD (**Cases C** and **D** in Figure 5.1). Note that the **Cases E** and **F** are handled outside of the three stages. Subroutines PARTITION, PAD, SWITCH and PROC1-3 are described in Figure 5.3. The subroutine PAD pads the final blocks of M and A if necessary, and computes j_M and j_A which help determine the final value of j for producing the tag.

```
procedure PARTITION($A, M$)
    $b \leftarrow n + m$
    $M_1 \| \dots \| M_\ell \overset{m}{\leftarrow} M$                ▷ $(\ell = |M|_m)$
    $A' \leftarrow \mathsf{left}_{(\ell+1)\cdot n}(A)$
    $A^* \leftarrow \mathsf{right}_{|A|-(\ell+1)\cdot n}(A)$
    $A'_1 \| \dots \| A'_{a'} \overset{n}{\leftarrow} A'$               ▷ $(a' = |A'|_n)$
    $A^*_1 \| \dots \| A^*_{a^*} \overset{b}{\leftarrow} A^*$          ▷ $(a^* = |A^*|_{n+m})$
end procedure
```

```
procedure PAD($A', A^*, M$)
    if $|M| \bmod m \neq 0$ then
        $\bar{M}_\ell \leftarrow M_\ell \| 10^{m-|M_\ell|-1}$
        $j_M \leftarrow 1$
    else
        $\bar{M}_\ell \leftarrow M_\ell$
        $j_M \leftarrow 0$
    end if
    if $|A'| \bmod n \neq 0$ then
        $A'_{a'} \leftarrow A'_{a'} \| 10^{n-|A'_{a'}|-1}$
        $j_A \leftarrow 2$
    else if $|A^*| \bmod n + m \neq 0$ then
        $A^*_{a^*} \leftarrow A^*_{a^*} \| 10^{n+m-|A^*_{a^*}|-1}$
        $j_A \leftarrow 2$
    else
        $j_A \leftarrow 0$
    end if
end procedure
```

```
procedure SWITCH($\Delta, j, j_{\text{new}}$)
    $\Delta \leftarrow \Delta \oplus L_*(\mathsf{int}(\langle j \rangle_4 \oplus \langle j_{\text{new}} \rangle_4))$
    $j \leftarrow j_{\text{new}}$
end procedure
```

```
procedure PROC1($M, A', H, \Delta, i$)
    for $r \leftarrow i$ to $a'$ do
        $C_{r-1} \leftarrow H \oplus M_{r-1}$
        $H \leftarrow H \oplus A'_r$
        $\Delta \leftarrow \Delta \oplus L[\mathtt{ntz}(r)]$
        $H \leftarrow F_K(H \oplus \Delta, M_{r-1})$
    end for
    $i \leftarrow a'$
end procedure
```

```
procedure PROC2($M, H, \Delta, i$)
    for $r \leftarrow i$ to $\ell$ do
        $C_{r-1} \leftarrow H \oplus M_{r-1}$
        $\Delta \leftarrow \Delta \oplus L[\mathtt{ntz}(r)]$
        $H \leftarrow F_K(H \oplus \Delta, M_{r-1})$
    end for
end procedure
```

```
procedure PROC3($A^*, H, \Delta, i$)
    for $r \leftarrow 1$ to $a^* - 1$ do
        $H \leftarrow H \oplus \mathsf{left}_n(A^*_r)$
        $\Delta \leftarrow \Delta \oplus L[\mathtt{ntz}(i + r - 1)]$
        $H \leftarrow F_K(H \oplus \Delta, \mathsf{right}_m(A^*_r))$
    end for
end procedure
```

Figure 5.3 – **The subroutines of the encryption algorithm of p-OMD[$\boldsymbol{F}, \boldsymbol{\tau}$]** (see Figure 5.2).

Enc.: Partitioning the message and associated data. The inputs are partitioned by PARTITION subroutine described in Figure 5.3. Let $M_1 \| M_2 \cdots \| M_\ell \overset{m}{\leftarrow} M$. Let $A' \| A^* \leftarrow A$ where $A' \leftarrow \mathsf{left}_{(\ell+1)\cdot n}(A)$ and $A^* \leftarrow \mathsf{right}_{|A|-(\ell+1)\cdot n}(A)$.

Let $A'_1 \| A'_2 \cdots \| A'_{a'} \overset{n}{\leftarrow} A'$ and $A^*_1 \| A^*_2 \cdots \| A^*_{a^*} \xleftarrow{n+m} A^*$. The string A' consists of $a' \leq \ell + 1$ n-bit blocks and these blocks will be simply absorbed into the chaining variable during the message encryption. In a typical use case where the associated data is shorter than the message, we will have $A' = A$ and $A^* = \varepsilon$ (**Case A** and **Case B** in Figure 5.1). The string A^* will be non-empty only if $|A| > (\ell + 1)n$, in which case, while A^* is being processed, there are no more message blocks to encrypt. To maximize the efficiency, we partition the string A^* into $n + m$-bit blocks so that we can make use of the whole input to F (see **Case C** and **Case D** in Figure 5.1).

Enc.: Processing the message and associated data. The message and associated data blocks are processed by the modified Merkle-Damgård iteration of F as shown in Figure 5.1. For every call to F, the n-bit input (chaining variable) is masked by the value $\Delta_{N,i,j}$, where i starts with the value $i = 1$ at the first call to F and is incremented for every call, and the j component is used to separate logical parts in the encryption process, as well as different types of input arguments. An appropriate use of the j component is essential for security.

Enc.: Selection of the j component in the index of $\Delta_{N,i,j}$. We use several values of j to separate the calls to the masked F in different contexts. We classify the calls to the masked F to two types: (1) the final call to F which returns the tag, and (2) the remaining internal calls. We note that in the special case that $M = \varepsilon$ and $|A| \leq n$ there will be only one call to F which returns the tag; hence, it is considered as the final call.

Internal Calls. We use $j \in \{0, 1, 2\}$ for the internal F-calls as follows.

For $i = 1$, i.e. the first call to F, the value of j is determined like this:

- if $\ell > 0$ and $a' > 0$ then let $j = 0$,
- if $\ell > 0$ and $a' = 0$ then let $j = 1$,
- if $\ell = 0$ and $a^* > 0$ then let $j = 2$.

For $1 < i < \ell + 1 + a^*$, depending on the presence of message blocks and AD blocks to be processed at the i^{th} call to the masked F, we have:

- if both an n-bit AD block and an m-bit message block are present then $j = 0$,
- if only an m-bit message block is present (no AD block is processed) then $j = 1$,
- if only an $(n+m)$-bit AD block is present (no message block is processed) then $j = 2$.

Final Call. The final call to F, which produces the authentication, tag uses $j_f \in \{3, 4, 5, \ldots, 14, 15\}$. If the tag is produced by a call to F with $i \neq 1$, we have three main cases depending on the inputs to the final masked F.

- If both an AD block and a message block are present in the final call (see **Case A** in Figure 5.1) then $j_f \in \{4, 5, 6, 7\}$; we let $j_f = 4$ if $|M_\ell| = m$ and $|A'_{a'}| = n$; let $j_f = 5$ if $|M_\ell| < m$ and $|A'_{a'}| = n$; let $j_f = 6$ if $|M_\ell| = m$ and $|A'_{a'}| < n$, and otherwise ($|M_\ell| < m$ and $|A'_{a'}| < n$) let $j_f = 7$.
- If only a message block is present but no AD block is processed in the final call (see **Case B** in Figure 5.1) then $j_f \in \{8, 9, 10, 11\}$; we let $j_f = 8$ if $|M_\ell| = m$ and $|A'_{a'}| = n$; let $j_f = 9$ if $|M_\ell| < m$ and $|A'_{a'}| = n$; let $j_f = 10$ if $|M_\ell| = m$ and $|A'_{a'}| < n$, and otherwise let $j_f = 11$ if $|M_\ell| < m$ and $|A'_{a'}| < n$. For the special case where there is no associate data at all, i.e. $A = \varepsilon$, we let $j_f = 8$ if $|M_\ell| = m$ and let $j_f = 9$ if $|M_\ell| < m$.
- If only an AD block is present but no message block is processed in the final call (see **Case C** and **Case D** in Figure 5.1) then $j_f \in \{12, 13, 14, 15\}$;

we let $j_f = 12$ if $|M_\ell| = m$ and $|A^*_{a^*}| = n + m$; let $j_f = 13$ if $|M_\ell| < m$ and $|A^*_{a^*}| = n + m$; let $j_f = 14$ if $|M_\ell| = m$ and $|A^*_{a^*}| < n + m$, and otherwise let $j_f = 15$ if $|M_\ell| < m$ and $|A^*_{a^*}| < n+m$. For the special case where there is no message at all, i.e. $M = \varepsilon$, let $j_f = 12$ if $|A^*_{a^*}| = n+m$ and let $j_f = 14$ if $|A^*_{a^*}| < n + m$.

For $i = 1$ (meaning that the final call is the same as the first call, which happens if $M = \varepsilon$ AND $|A| \leq n$) we need to apply a special treatment:

- if both $M = A = \varepsilon$ then $j_f = 3$ (**Case F** in Figure 5.1),
- if $M = \varepsilon$ and $0 < |A| \leq n$ then we let $j_f = 12$ if $|A| = n$, otherwise, let $j_f = 14$ (**Case E** box in Figure 5.1).

Note that there is no variable j_f in Figure 5.2 as j_f corresponds to a special use of variable j in the last call to F. Specifically, j_f corresponds to the calls to the SWITCH subroutine that use the value of new j of the form $\mathsf{const} + j_A + j_M$ or the value 3.

Decryption algorithm. Considering that the encryption process of p-OMD is actually a self-synchronizing stream cipher with an integrated authentication mechanism, the decryption process proceeds in a very similar way as the encryption process up until the verification of the tag, which happens at the end of the decryption process where the newly computed tag T' is compared with the provided tag T. If $T' = T$ then output M, otherwise output $\perp$.

5.5 Security Analysis

The security analysis for p-OMD is modular and follows similar steps as the analysis of OMD:

Step 1: We first analyse the security of a generalized variant of p-OMD$[F, \tau]$ where the masked F gets replaced by an ideal primitive; namely, a (tweakable) random function $\widetilde{R}$. This generalized scheme is called p-$\mathbb{OMD}[\widetilde{R}, \tau]$ and is illustrated in Figure 5.4. This is the major proof step which differs from and is more involved than that of OMD.

Step 2: We instantiate the (tweakable) random function $\widetilde{R}$ by a (tweakble) PRF $\widetilde{F}$. This is a standard step and it is the same as for OMD.

Step 3: We instantiate a (tweakable) PRF using a PRF (with a smaller domain) using the XE method [Rog04a] with the masking sequence based on the canonical Gray code [RBBK01, KR11]. This step is similar to that of OMD, only the details of the mask generation function differ.

The security bound for p-OMD is stated in Theorem 5.1. It is interesting to note that the security bound is the same as that of OMD, showing that the modifications we made to OMD to obtain p-OMD are without any loss of security.

Theorem 5.1. *Fix $n \geq 1$, $0 \leq \tau \leq n$. Let $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be a PRF with $1 \leq m \leq n$. Let $\mathscr{A}$ be a CPA adversary that runs in time t, makes q_e encryption queries that induce no more than σ_e n calls to F in total, such that no individual input (AD or message) is more that ℓ_{max} m-bit blocks long. Let further $\mathscr{A}'$ be a CCA adversary that runs in time t', makes q'_e encryption queries and q'_d decryption queries that induce no more than σ' calls to F in total, such that no individual input is more that ℓ'_{max} m-bit blocks long. We have*

$$\begin{aligned} \mathbf{Adv}^{\mathbf{priv}}_{\mathrm{p-OMD}[F,\tau]}(\mathscr{A}) &\leq \mathbf{Adv}^{\mathbf{prf}}_{F}(\mathscr{B}) + \frac{3\sigma_e^2}{2^n} \\ \mathbf{Adv}^{\mathbf{auth}}_{\mathrm{p-OMD}[F,\tau]}(\mathscr{A}') &\leq \mathbf{Adv}^{\mathbf{prf}}_{F}(\mathscr{B}') + \frac{3\sigma^2}{2^n} + \frac{q_d \ell_{max}}{2^n} + \frac{q_d}{2^\tau} \end{aligned}$$

for some $\mathscr{B}$ that makes $2 \cdot \sigma_e$ queries and runs in time $t + \gamma \cdot n \cdot \sigma$ for some constant γ, and $\mathscr{B}$' that makes $2 \cdot \sigma'$ queries and runs in time $t' + \gamma' \cdot n \cdot \sigma'$ for some constant γ'.

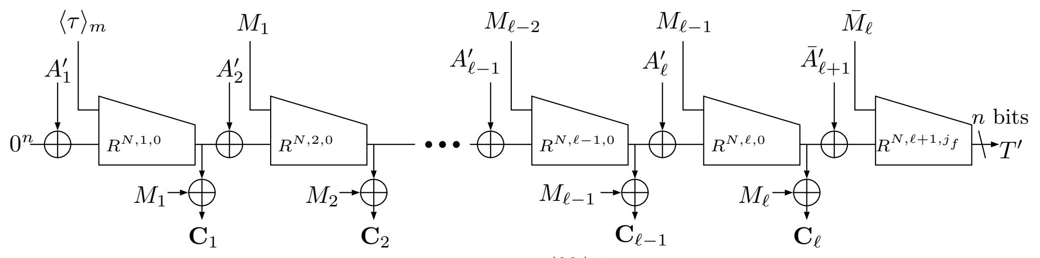
Proof. The proof is obtained by combining Lemma 5.2 with Lemma 5.3 and Lemma 5.4. □

Idealization of p-OMD. The scheme p-$\mathbb{OMD}[\widetilde{R},\tau]$ is a generalization of p-OMD$[F,\tau]$ that uses a (tweakable) random function $\widetilde{R} \leftarrow\$ \widetilde{\mathrm{Func}}(\mathcal{T}, 2^{m+n}, 2^n)$ instead of the masked F. The p-$\mathbb{OMD}[\widetilde{R},\tau]$ is depicted in Figure 5.4. The tweak space $\mathcal{T}$ consists of sixteen mutually exclusive sets of tweaks $\mathcal{T} = \bigcup_{j=0}^{15} \mathcal{N} \times \mathbb{N} \times \{j\}$.

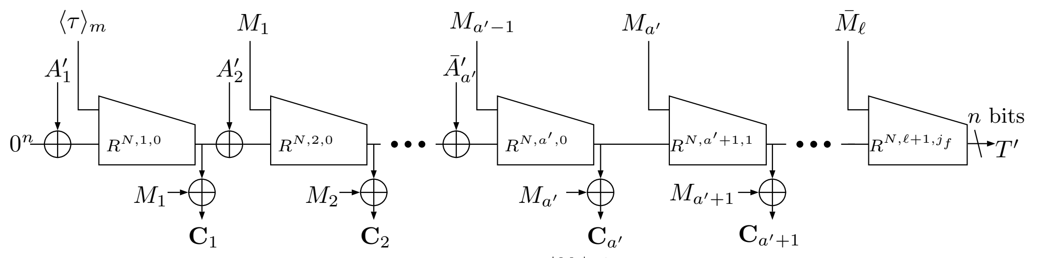
Lemma 5.2. *Let p-$\mathbb{OMD}[\widetilde{R},\tau]$ be the scheme shown in Figure 5.4. Let $\mathscr{A}$ be an information-theoretic CPA adversary that makes q_e encryption queries that induce no more than σ_e calls to $\widetilde{R}$ in total, such that no individual input (AD or message) is more that ℓ_{max} m-bit blocks long. Let further $\mathscr{A}'$ be an information theoretic CCA adversary that makes q'_e encryption queries and q'_d decryption queries that induce no more than σ' calls to $\widetilde{R}$ in total, such that no individual input is more that ℓ'_{max} m-bit blocks long. Then*

$$\begin{aligned} \mathbf{Adv}^{\mathbf{priv}}_{p\text{-}\mathbb{OMD}[\widetilde{R},\tau]}(\mathscr{A}) &= 0 \\ \mathbf{Adv}^{\mathbf{auth}}_{p\text{-}\mathbb{OMD}[\widetilde{R},\tau]}(\mathscr{A}') &\leq \frac{q_d \ell_{max}}{2^n} + \frac{q_d}{2^\tau} \end{aligned}$$

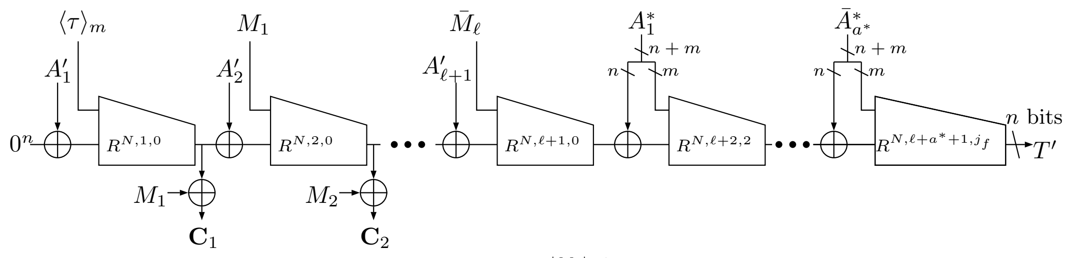
Proof. The proof of the PRIV bound is straightforward. Each of the encryption queries $(N^1, A^1, M^1) \cdots (N^{q_e}, A^{q_e}, M^{q_e})$ asked by $\mathscr{A}$ has a distinct nonce. Referring to Figure 5.4, this means that every evaluation of $\widetilde{R}^{N^x,i,j}(\cdot)$ uses a distinct tweak, hence the



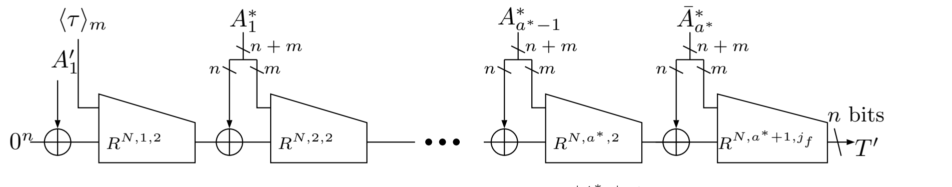
Case A: $\ell > 0$ and $|\mathbf{A}|_\mathbf{n} = \ell + 1$. Let $\bar{M}_\ell = M_\ell||10^{m-|M_\ell|-1}$ if $|M_\ell| < m$ and $\bar{M}_\ell = M_\ell$ otherwise. Let $\bar{A}'_{a'} = A'_{a'}||10^{n-|A'_{a'}|-1}$ if $|A'_{a'}| < n$ and $\bar{A}'_{a'} = A'_{a'}$ otherwise.



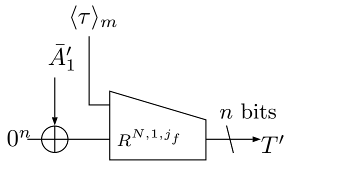
Case B: $\ell > 0$ and $|\mathbf{A}|_\mathbf{n} < \ell + 1$. Let $\bar{M}_\ell = M_\ell||10^{m-|M_\ell|-1}$ if $|M_\ell| < m$ and $\bar{M}_\ell = M_\ell$ otherwise. Let $\bar{A}'_{a'} = A'_{a'}||10^{n-|A'_{a'}|-1}$ if $|A'_{a'}| < n$ and $\bar{A}'_{a'} = A'_{a'}$ otherwise.



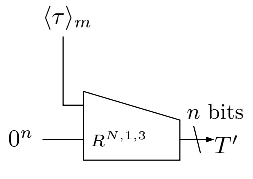
Case C: $\ell > 0$ and $|\mathbf{A}|_\mathbf{n} > \ell + 1$. Let $\bar{M}_\ell = M_\ell||10^{m-|M_\ell|-1}$ if $|M_\ell| < m$ and $\bar{M}_\ell = M_\ell$ otherwise. Let $\bar{A}^*_{a^*} = A^*_{a^*}||10^{n+m-|A^*_{a^*}|-1}$ if $|A^*_{a^*}| < n + m$ and $\bar{A}^*_{a^*} = A^*_{a^*}$ otherwise.



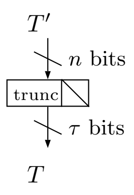
Case D: $\mathbf{M} = \varepsilon$ and $|\mathbf{A}| > \mathbf{n}$. Let $1\bar{A}^*_{a^*} = A^*_{a^*}||10^{n+m-|A^*_{a^*}|-1}$ if $|A^*_{a^*}| < n + m$ and $\bar{A}^*_{a^*} = A^*_{a^*}$ otherwise.



Case E: $\mathbf{M} = \varepsilon$ and $0 < |\mathbf{A}| \leq \mathbf{n}$. Let $\bar{A}'_1 = A'_1||10^{n-|A'_1|-1}$ if $|A'_1| < n$ and $\bar{A}'_1 = A'_1$ otherwise.



Case F
$\mathbf{M} = \mathbf{A} = \varepsilon$



Obtaining the final tag.

Figure 5.4 – **The p-$\mathbb{OMD}[\widetilde{R}, \tau]$ scheme** using a tweakable random function $\widetilde{R} \leftarrow_\$ \widetilde{\text{Func}}(\mathcal{T}, 2^{n+m}, 2^n)$.

ciphertexts $\mathbb{C}^r$ for $1 \leq r \leq q_e$ that the adversary sees are uniform and independent.

The proof of the authenticity bound requires a rather involved case analysis. A visualisation of the hierarchy of the cases as a tree is presented in Figure 5.5 to improve the clarity of the proof. We first analyse the case where the adversary makes a single verification query and then we use the generic result of Bellare et al. [BGM04] to get a bound against adversaries that make multiple verification queries.

Let $\mathscr{A}'$ be an adversary making q_e encryption queries $(N^1, A^1, M^1) \cdots (N^{q_e}, A^{q_e}, M^{q_e})$ and a single decryption query $(N, A, \mathbb{C})$.

Let $M^i = M_1^i \cdots M_{\ell_i}^i$ be the message and $A^i = A'^i_1 \cdots A'^i_{a'_i} || A^{*i}_1 \cdots A^{*\,i}_{a^*_i}$ be the associated data in the ith encryption query. Let $\mathbb{C}^i = C^i || T^i$ be the ciphertext received for query (N^i, A^i, M^i). We let $x_i = 1 + \ell_i + a_i^*$ denote the number of calls to the (tweakable) random function $\widetilde{R}$ made while processing the ith query. Note that x_i is also the value of the second tweak component used in the final call to the compression function which produces T^i. We further let j_f^i denote the third component of the tweak used in the final call to $\widetilde{R}$ when processing the ith query.

Let $A = A'_1 \cdots A'_{a'} || A^*_1 \cdots A^*_{a^*}$ be the associated data, $\mathbb{C} = C || T$ the ciphertext where $C = C_1 \cdots C_\ell$ and $T \in \{0,1\}^\tau$ be the tag in the decryption query. Let $M = M_1 \cdots M_\ell$ denote the corresponding decrypted message. We let $x = 1 + \ell + a^*$ be the number of calls to $\widetilde{R}$ made while processing the forgery attempt. This is also the value of the second tweak component in the final call to the compression function that is supposed to produce the T. We further let j_f denote the third component of the tweak used in the final call to $\widetilde{R}$ when processing the alleged forgery.

In the proof we use the intermediate chaining variables that occur in the query processing. We let H_r^i denote the *output* of the rth call to the compression function in the processing of the ith encryption query, so we have $H_1^i = \widetilde{R}^{N,1,0}(A'^i_1, \langle \tau \rangle_m)$ and $T^i = \mathsf{left}_\tau \left(H^i_{x_i} \right)$. Similarly, we let H_r stand for rth intermediate chaining value in the processing of the forgery attempt.

We have the following disjoint cases that can occur upon the decryption query of $\mathscr{A}'$. The negation of an event E is denoted as $\bar{\mathsf{E}}$.

Case 1: $N \notin \{N^1, \cdots N^{q_e}\}$. We let $\mathsf{E_1}$ denote the event $N \notin \{N^1, \cdots N^{q_e}\}$ in the following. The adversary has to find a correct T that is the first τ bits produced by a call to $\widetilde{R}^{N,x,j_f}(\cdot)$. Because the nonce-component N of the tweak N, x, j_f has not been used in any encryption query, $\mathscr{A}'$ has not seen any image under $\widetilde{R}^{N,x,j_f}(\cdot)$ Thus the probability that the adversary can succeed in finding correct value of T is $2^{-\tau}$.

All the following cases are conditioned by $\bar{\mathsf{E}}_1$, the negation of $\mathsf{E_1}$. That is, $N = N^i$ for a *single* $i \in \{1, \ldots, q_e\}$ (noticing that no nonce is reused during encryption queries). We can ignore all other than the ith query since the corresponding ciphertexts are statistically independent with the images of $\widetilde{R}$ used to process the forgery attempt $N, A, \mathbb{C}$ with $N = N^i$.

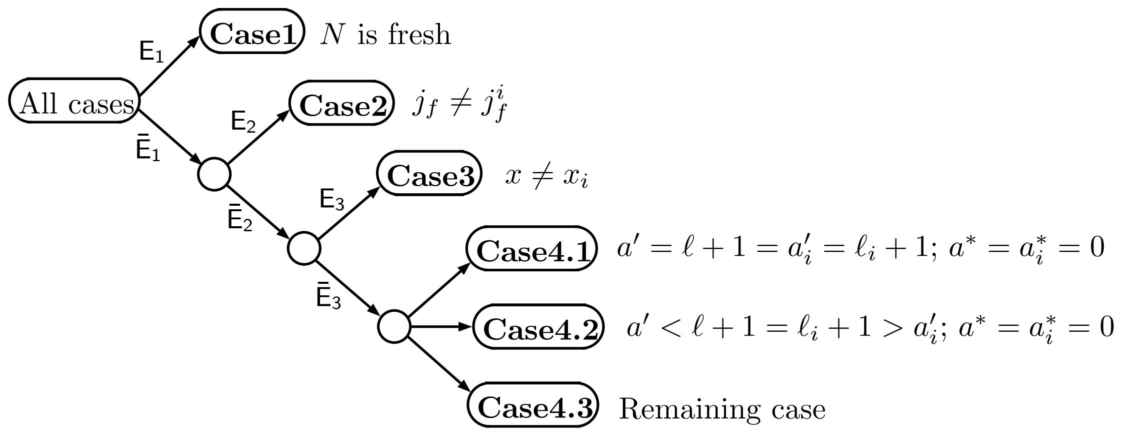


Figure 5.5 – **The structure of the proof of p-OMD's authenticity.** A condition on an edge applies to the whole subtree.

Case 2: $\bar{\mathsf{E}}_1 \wedge \mathsf{E}_2$, where E_2 is the event that $j_f \neq j_f^i$. Recall that a successfully forged T must be the first τ bits of a value produced by $\widetilde{R}^{N,x,j_f}(\cdot)$. The inequality $j_f \neq j_f^i$ occurring in this case implies the adversary has not seen any image under $\widetilde{R}^{N,x,j_f}(\cdot)$ (no matter what are the values of x and x_i) and the adversary has to guess the correct T. The probability of a successful forgery is therefore $2^{-\tau}$.

We introduce auxiliary notation for the analysis of the remaining cases. Consider the i^{th} encryption query. Depending on the length of the message $|M^i|$ and the length of AD $|A^i|$, we can have three situations. In the *first situation*, we have $|M^i|_m + 1 = |A^i|_n$ and $|M^i| > 0$ (**Case A** in Figure 5.4). This means that the compression function call used to produce T^i has a block of message at its m-bit input and an AD block xored to the chaining variable at its n-bit input. We denote this event as $\texttt{type-1}^i$ and we note that $j_f^i \in \{4, 5, 6, 7\}$. The *second possible situation* arises if $|M^i|_m + 1 > |A^i|_n$ with $|M^i| > 0$ (**Case B** in Figure 5.4). There is no block of the AD xored to the n-bit input of the final compression function call. We denote this event as $\texttt{type-2}^i$ and we note that $j_f^i \in \{8, 9, 10, 11\}$. The *last possible situation* is when either $|M^i|_m + 1 < |A^i|_n$ and $|A^i| > n$ so there is a block of AD xored to the n-bit input as well as another block fed directly to the m-bit input in the final call to the compression function (**Cases C, D** in Figure 5.4) or $0 < |A^i| \leq n$ and $M^i = \varepsilon$ (**Case E** in Figure 5.4). We denote this by $\texttt{type-3}^i$ and we note that $j_f^i \in \{12, 13, 14, 15\}$.

We define `type-1`, `type-2` and `type-3` for the forgery attempt in a similar way (note that $|C| = |M|$).

In the following, we need to address the event $\bar{\mathsf{E}}_1 \wedge \bar{\mathsf{E}}_2$ i.e., $N = N^i$, $j_f = j_f^i$. We remark that the condition $\bar{\mathsf{E}}_2$ is met for a valid forgery (i.e., $(A, M) \neq (A^i, M^i)$) if and only if *both* the i^{th} encryption query and the alleged forgery are

- non-empty, i.e., $(A, M) \neq (\varepsilon, \varepsilon) \wedge (A^i, M^i) \neq (\varepsilon, \varepsilon)$,

- padded in the same way, i.e.,

$$\begin{aligned}&(|C| \equiv 0 \pmod m \Leftrightarrow |C^i| \equiv 0 \pmod m)\\ \text{and } &(|A'| \equiv 0 \pmod n \Leftrightarrow |{A'}^i| \equiv 0 \pmod m)\\ \text{and } &(|A^*| \equiv 0 \pmod{m+n} \Leftrightarrow |{A^*}^i| \equiv 0 \pmod{m+n})\end{aligned}$$

  (in other words, we pad the last block of M iff we pad the last block of M^i and the same applies for associated data),

- of the same "type", i.e.,

$$\left(\texttt{type-1} \wedge \texttt{type-1}^i\right) \vee \left(\texttt{type-2} \wedge \texttt{type-2}^i\right) \vee \left(\texttt{type-3} \wedge \texttt{type-3}^i\right).$$

Case 3: $\bar{\mathsf{E}}_1 \wedge \bar{\mathsf{E}}_2 \wedge \mathsf{E}_3$ where E_3 stands for the event that $x \neq x_i$. Recall that T^i is produced as the τ most significant bits of an image under $\widetilde{R}^{N,x_i,j^i_f}(\cdot)$ for some j^i_f, and T is produced as the τ most significant bits of an image under $\widetilde{R}^{N,x,j_f}(\cdot)$ for some j_f. We have two sub-cases.

Case 3a: If $x > x_i$ then $\mathscr{A}'$ has seen no image under $\widetilde{R}^{N,x,j_f}(\cdot)$ regardless of the value of j_f and the probability of a successful forgery (equivalent to guessing τ random bits) is $2^{-\tau}$.

Case3b: If $x < x_i$ then a single image under $\widetilde{R}^{N,x,j^i}(\cdot)$ was used in processing of the i^{th} encryption query. However T is produced by $\widetilde{R}^{N,x,j_f}(\cdot)$, such that $j^i \neq j_f$ because the values of j_i used for "internal" calls to $\widetilde{R}$ are disjoint with those that produce tags. The probability of a successful forgery (equivalent to guessing τ random bits) is $2^{-\tau}$.

Case 4: It remains to address the cases, where we have $\bar{\mathsf{E}}_1 \wedge \bar{\mathsf{E}}_2 \wedge \bar{\mathsf{E}}_3$, i.e., the case, when the i^{th} encryption query and the alleged forgery (1) share the same nonce, (2) are padded in the same way, are both non-empty, are of the same "type" so $j_f = j^i_f$ and (3) they are both processed with the same number of calls to the compression function. We investigate each of the three possible "types" separately.

Case 4.1: $\bar{\mathsf{E}}_1 \wedge \bar{\mathsf{E}}_2 \wedge \bar{\mathsf{E}}_3 \wedge \left(\texttt{type-1} \wedge \texttt{type-1}^i\right)$. This means that $a' = \ell + 1 = x = x_i = {a'}_i = \ell_i + 1$, $a^* = 0$ and ${a^*}_i = 0$. Both T^i and T are produced by the same RF $\widetilde{R}^{N,x,4}(\cdot)$. W.l.o.g. assume that both $|A'|$ and $|{A'}^i|$ are a multiple of n and both $|M|$ and $|M^i|$ are a multiple of m. We can make this assumption because $\bar{\mathsf{E}}_2$ holds and because in the other cases the incomplete blocks are injectively padded to full length.

The adversary can succeed in producing a valid forgery in two ways. Either the inputs into the last $\widetilde{R}$-call in the processing of the i^{th} encryption query and of the forgery attempt are distinct, i.e., $(H^i_{x-1} \oplus {A'}^i_{a'}, M^i_\ell) \neq (H_{x-1} \oplus {A'}_{a'}, M_\ell)$, or they are equal, i.e., $(H^i_{x-1} \oplus {A'}^i_{a'}, M^i_\ell) = (H_{x-1} \oplus {A'}_{a'}, M_\ell)$. In the former case, the adversary is left with the task of guessing the output value of a RF on an input, that has not been evaluated before which is bounded with the probability $p_{fn} = 2^{-\tau}$.

In the latter case, the equality $(H^i_{x-1} \oplus A'^i_{a'}, M^i_\ell) = (H_{x-1} \oplus A'_{a'}, M_\ell)$ permits the adversary to set $T = T^i$. We must have $(N, A, M) \neq (N^i, A^i, M^i)$, so there is a position r in which the two queries differ, i.e., we must have an $1 \leq r < x$, such that $(A'_r, M_r) \neq (A'^i_r, M^i_r)$ and after which the queries are identical. So $\mathscr{A}'$ has surely not seen the image $H_r = \widetilde{R}^{N,r,j_r}(H_{r-1} \oplus A'_r, M_r)$ but he must ensure that $H_r = H^i_r$. This happens with a probability of 2^{-n} for a single r. We bound the total probability of achieving the final collision by $p_{fe} = (x-1)2^{-n}$ obtained as a union bound over r.

The bound of Case 4.1 is finally obtained as the sum

$$p_{fn} + p_{fe} = (x-1)2^{-n} + 2^{-\tau} \leq \ell_{max} \cdot 2^{-n} + 2^{-\tau}.$$

Case 4.2: $\bar{\mathsf{E}}_1 \wedge \bar{\mathsf{E}}_2 \wedge \bar{\mathsf{E}}_3 \wedge (\texttt{type-2} \wedge \texttt{type-2}^i)$. This implies $\ell + 1 > |A|_n$, $\ell_i + 1 > |A^i|_n$ and $a^* = a^*{}_i = 0$, so $x = x' = \ell + 1 = \ell_i + 1$. W.l.o.g. assume that both $|A'|$ and $|A'^i|$ are a multiple of n and both $|M|$ and $|M^i|$ are a multiple of m by similar argument as in Case 4.1. We have two subcases:

Case 4.2a: $|A|_n = |A^i|_n$, i.e., $a' = a'_i$. Analysis of this case is very similar to Case 4.1. Again we observe, that the adversary's chance to produce a forgery is bounded by $2^{-\tau}$ if the inputs to the final RF are distinct. The adversary can reuse T^i if he manages to force the collision on the inputs to the final RF. The probability that $\mathscr{A}'$ can succeed in forcing this collision is bounded in the same way as in Case 4.1 by summing 2^{-n} for $1 \leq r < x$. For $r > a'$, we have no more blocks of A' to consider, which gives the adversary even less power. We conclude that the probability of forgery in this case is bounded by $2^{-\tau} + \ell_{max} \cdot 2^{-n}$.

Case 4.2b: $|A|_n \neq |A^i|_n$, i.e., $a' \neq a'_i$. The analysis of this case is very similar to the previous one. We need to additionally consider that if $a' > a'_i$ then there is at least one r, such that $1 \leq r < x$ and there is a block A'_r but there is no block A'^i_r (or the other way around if $a' < a'_i$). This implies that $\mathscr{A}'$ may be able to force the same data inputs to be fed to the r^{th} call to $\widetilde{R}$. However, these will processed with different tweaks $\widetilde{R}^{N,r,0}(\cdot)$ and $\widetilde{R}^{N,r,1}(\cdot)$, ensuring that the collision on H_r happens with probability 2^{-n}. Keeping this in mind, the analysis of this case follows the same structure as the previous case and we conclude that the probability of forgery is bounded by $2^{-\tau} + \ell_{max} \cdot 2^{-n}$

Case 4.3: $\bar{\mathsf{E}}_1 \wedge \bar{\mathsf{E}}_2 \wedge \bar{\mathsf{E}}_3 \wedge (\texttt{type-3} \wedge \texttt{type-3}^i)$ so $\ell + 1 < |A|_n$ and $\ell_i + 1 < |A^i|_n$, so $x = x' = \ell + 1 + a^* = \ell_i + 1 + a^*{}_i$. W.l.o.g. assume that both $|A^*|$ and $|A^{*i}|$ are a multiple of $n+m$ and both $|M|$ and $|M^i|$ are a multiple of m by similar argument as in Case 4.1. We have three subcases.

Case 4.3a: $0 < |A| \leq n$, $0 < |A^i| \leq n$ and $M = M^i = \varepsilon$. W.l.o.g. assume $|A| = |A^i| = n$ (due to injective padding). Since the alleged forgery must be different from all encryption queries, we have $A \neq A^i$ and the adversary must guess the output of a RF on a new input. The probability of forgery is thus $2^{-\tau}$. In following two subcases we have $|A| > n$ and $|A^i| > n$.

Case 4.3b: $|M|_m = |M^i|_m$, i.e., $\ell = \ell_i$ and $a^* = {a^*}_i$. The analysis is almost identical as in case 4.2a, with the difference that for $r > \ell + 1$ we have no more blocks of M to consider but we have $n+m$ blocks of AD instead. The probability of inner collisions $H_r = H_r^i$ for $r > \ell + 1$ is thus also 2^{-n} and we conclude that the probability of forgery is bounded by $2^{-\tau} + \ell_{max} \cdot 2^{-n}$.

Case 4.3c: $|M|_m \neq |M^i|_m$, i.e., $\ell \neq \ell_i$ and $a^* \neq {a^*}_i$. Similarly as in Case 4.2b we need to take into account that the adversary can change the length of message and AD, so that there must be r such that $1 \leq r < x$ and such that there is M_r but no M_r^i (or the other way around). The domain separation by the j-component of the tweaks again ensures that the probability of internal collision $H_r = H_r^i$ is 2^{-n} for such r. We conclude that the probability of forgery is bounded by $2^{-\tau} + \ell_{max} \cdot 2^{-n}$.

Finally, using the results of Bellare et al. [BGM04] we get the bound against adversaries that make q_d decryption queries as $\frac{q_d}{2^\tau} + \frac{q_d \ell_{max}}{2^n}$. □

Instantiation of Tweakable RFs with Tweakable PRFs. This is a classical step in which the ideal primitive—tweakable random function $\widetilde{R}$—is replaced with a standard primitive—tweakable PRF $\widetilde{F}$. The security loss induced by this step is stated in the following lemma.

Lemma 5.3. *Let $\widetilde{R} : \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \rightarrow \{0,1\}^n$ be a RF and $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \rightarrow \{0,1\}^n$ be a keyed function. Let $\mathscr{A}$ be a CPA adversary that runs in time t, makes q_e encryption queries that induce no more than σ_e n calls to F in total, such that no individual input (AD or message) is more that ℓ_{max} blocks long. Let further $\mathscr{A}'$ be a CCA adversary that runs in time t', makes q_e' encryption queries and q_d' decryption queries that induce no more than σ' calls to F in total, such that no individual input is more that ℓ'_{max} blocks long. Then*

$$\begin{aligned}
\mathbf{Adv}^{\mathbf{priv}}_{p\text{-}\mathbb{OMD}[\widetilde{F},\tau]}(\mathscr{A}) &\leq \mathbf{Adv}^{\mathbf{priv}}_{p\text{-}\mathbb{OMD}[\widetilde{R},\tau]}(\mathscr{B}) + \mathbf{Adv}^{\mathbf{prf}}_{\widetilde{F}}(\mathscr{C}) \\
\mathbf{Adv}^{\mathbf{auth}}_{p\text{-}\mathbb{OMD}[\widetilde{F},\tau]}(\mathscr{A}') &\leq \mathbf{Adv}^{\mathbf{auth}}_{p\text{-}\mathbb{OMD}[\widetilde{R},\tau]}(\mathscr{B}') + \mathbf{Adv}^{\mathbf{prf}}_{\widetilde{F}}(\mathscr{C}')
\end{aligned}$$

$\mathscr{B}$ and $\mathscr{B}'$ are information theoretic adversaries that have the same resources (except for time complexity) as $\mathscr{A}$ and $\mathscr{A}'$ respectively, and where $\mathscr{C}$ and $\mathscr{C}'$ make no more than σ and σ' queries respectively and run in time $t + \gamma \cdot n \cdot \sigma_e$ and $t' + \gamma' \cdot n \cdot \sigma'$ respectively with some constants γ, γ'.

Proof. The proof of this lemma uses the same reductions as the proof of Lemma 3.3. □

Instantiation of Tweakable PRFs with PRFs. The last step is to instantiate the (tweakable) PRFs by means of a (keyed) compression function which is assumed to be a secure PRF. We use a very similar technique to what is done in OMD and MR-OMD, inspired by the XE construction [Rog04a].

The difference between the OMD and p-OMD is the way that the masking sequence $\Delta_{N,i,j}$ is computed. This difference is introduced because we need more values of the j-component compared to OMD. In p-OMD, the tweak space is of the form

$$\mathcal{T} = \mathcal{N} \times \{0, 1, \ldots, 2^{n-6}\} \times \{0, 1, \cdots, 15\}.$$

Lemma 5.4. *Let $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be a keyed function key space $\mathcal{K}$. Let $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be defined by $\widetilde{F}_K^{\langle \mathsf{T} \rangle}(X,Y) = F_K((X \oplus \Delta_K(\mathsf{T})), Y)$ for every $\mathsf{T} \in \mathcal{T}, K \in \mathcal{K}$, $X \in \{0,1\}^n, Y \in \{0,1\}^m$ and let $\Delta_K(\mathsf{T})$ be the masking function of p-OMD as defined in Section 5.4. Let further $\mathscr{A}$ be an adversary that runs in time t and makes q queries. Then we have*

$$\mathbf{Adv}_{\widetilde{F}}^{\mathbf{prf}}(\mathscr{A}) \leq \mathbf{Adv}_{F}^{\mathbf{prf}}(\mathscr{B}) + \frac{3q^2}{2^n}$$

for some $\mathscr{B}$ that runs in time $t + \gamma \cdot q$ for a constant γ and makes $2q$ queries.

Proof. The proof of this lemma is an adaptation of the analysis made by Krovetz and Rogaway [KR11], by similar arguments as in the proof of Lemma 3.4.

For the proof to apply, we need to prove that $\Delta_K(\mathsf{T})$ is a 2^{-n}-uniform 2^{-n}-AXU hash. As shown in Appendix A.2, this can be easily verified.

□

5.6 Performance

One of the goals we sought to achieve with the design of p-OMD was to (significantly) improve over the performance of OMD. In this section, we investigate what is the exact improvement.

Predicting the speed-up. The speed-up of p-OMD over the original OMD (see Section 3.4) comes from the fact, that in most applications, we can dispense with all compression function calls that needed to be made by OMD to process AD.

Of course, this has to be compensated by a more complicated masking scheme, and (much) more complicated algorithm that makes sure correct masks are being computed. However, the computational cost of the saved compression function calls largely outweighs the overhead induced by the complicated masking scheme.

We can derive a theoretical upper-bound on the speed-up achieved by p-OMD relative to OMD. To simplify the analysis, we neglect the overhead induced by the masking, and assume the whole algorithm is executed sequentially (i.e. we do not consider parallelism), and we assume all L_* and L values are precomputed.

Given an encryption query (N, A, M), OMD needs to make $a + \ell + 2$ calls to the compression function, where $a = |A|_{m+n}$, $\ell = |M|_m$ and the two extra calls are needed to compute the tag and the initial N-dependent value. For the same query, p-OMD needs $\ell + a' + a^* + 2$ calls to the compression function, where $\ell = |M|_m$, $a' = \min(\ell + 1, |A|_n)$, $a^* = |\mathsf{right}_{|A|-a'\cdot n}(A)|_{m+n}$ and the 2 extra calls are the same as for OMD.

In most situations AD will not be too long, and we will have $a^* = 0$. Based on this assumption, we can estimate the relative speed-up of p-OMD over OMD as

$$S = \frac{2 + \ell + a}{2 + \ell},$$

i.e. p-OMD will be about S times faster than OMD.

We further simplify the analysis by assuming $n = m$ (as is the case for the compression functions of SHA256 and SHA512). The speed-up will be maximal if $a = \lfloor(\ell + 1)/2\rfloor$. In that case, we have

$$S = \frac{2 + \ell + a}{2 + \ell} \approx 1.5$$

if ℓ is large enough. Thus, we expect that under ideal conditions, p-OMD can be up to 1.5 times faster than OMD.

Experiments. To verify the predicted speed-up of p-OMD over OMD, we implemented the two algorithms in software and made measurements to determine and compare their performance.

The comparison is performed on the x86-64 architecture (Intel Core i7-3632QM, with all measurements carried out on a single core). For OMD, we used the OMD-sha512 instantiation optimised for the AVX1 instruction extension, which achieves the best performance for OMD (see Section 3.7). We made the necessary modifications (as in description of p-OMD) to the same code to obtain an implementation of p-OMD. Both OMD and p-OMD were instantiated with the same parameters: key length=512, nonce length=256, tag length=256. Both implementations have been built using the gcc compiler and setting the `-Ofast` optimization flag.

We measure the time complexity of the encryption process for varying lengths of message and associated data. For the rest of this section, let m denote the message length and a the AD length in bytes. We measure the encryption time for $m \in \{64, 128, 192, \ldots, 4096\}$ and $a \in \{64, 128, \ldots m\}$ for every value of m. That is, we consider the typical case when AD is at most as long as the message.

For both OMD and p-OMD and for every pair of values m, a, we measure the time of one encryption using the `rdtsc` instruction 200 times to compute the mean time. This is repeated 91 times and the value we take as the result is the median of these 91 mean encryption times. We additionally apply the same procedure to measure time

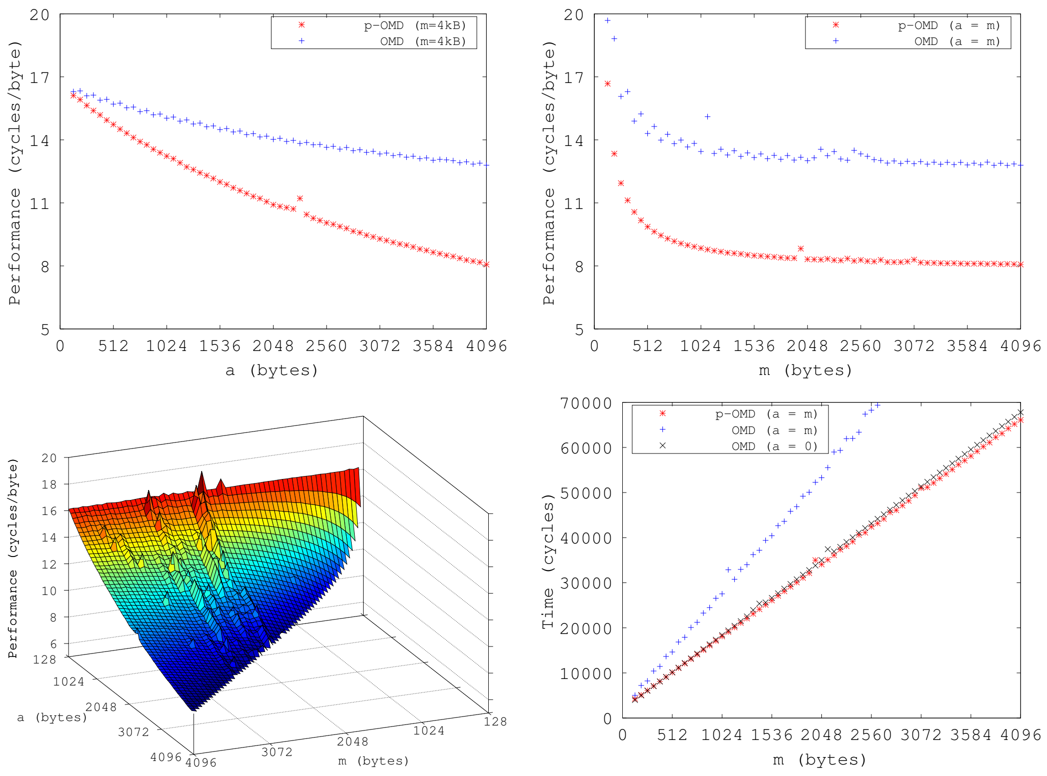


Figure 5.6 – **Performance comparisons between OMD and p-OMD. Top left**: encryption complexity with fixed message length. **Top right**: encryption complexity with equal message length and AD length. **Bottom right**: comparison of OMD without AD to OMD and p-OMD with AD. **Bottom left**: encryption complexity of p-OMD for varying message and AD lengths.

complexity of the encryption of OMD with $m \in \{64, 128, \ldots, 4096\}$ and $a = 0$. The results are shown in Figure 5.6.

The top left graph in Figure 5.6 shows that the relative complexity of encryption of both OMD and p-OMD decreases as the length of AD increases; however, p-OMD performs better than OMD. The top right graph demonstrates that *if the length of AD is close to the message length* then p-OMD has a clear advantage over OMD, and for longer messages we observe the speedup close to the predicted 150%. The bottom right graph confirms that the p-OMD provides an almost free authentication of associated data compared to OMD.

For both OMD and p-OMD, these measurements exclude the complexity of the precomputation step in computing $\Delta_{N,i,j}$ (see Section 5.4) which is done only once during the whole lifetime of a key. As an upper bound, we measure the complexity of the precomputation step that is sufficient to encrypt messages with length up to 2^{63} blocks. For OMD the precomputation step takes 5818 cycles while in p-OMD it requires 6863 cycles on average.

5.7 Follow-up Work

The main shortcoming of p-OMD is the complexity of the algorithm and the amount of precomputation it requires for an implementation to be truly efficient. The two major negative consequences of p-OMD's complexity are the difficulty of verifying the security proofs, and the difficulty of implementing the encryption and the decryption algorithms correctly, as pointed out by Ashur and Mennink [AM16].

They proposed an alternative called Spoed, [1] which reduces the complexity of the algorithms, but achieves slightly better quantitative security, and has the same computational complexity in most cases. Spoed uses an efficient padding function to injectively transform a message-AD pair into a sequence of $2n$-bit words, which then gets processed in a similar way to p-OMD's encryption algorithm. Because the padding scheme encodes the lengths of the two inputs into the output sequence, the encryption algorithm does not need to explicitly differentiate as many cases as p-OMD, and is simpler as a result.

Ashur and Mennink further propose Spoednic, which additionally multiples each half-block (of the output of the padding scheme) that gets xored to a chaining variable by a secret value. This additional masking lends Spoednic the ability to resist forgery attempts even in the nonce-misuse scenario.

[1]Which expands to "Simplified p-OMD encryption and decryption."

Chapter 6

Full-State Absorption for Sponge-based Constructions

In this chapter, we introduce two related modes of operation for a cryptographic permutation called *Full-state Keyed Sponge* (FKS) and *Full-state Keyed Duplex* (FKD). While these two are not constructions for authenticated encryption themselves, we show that highly efficient AE constructions can be built on top of the FKD.

The work presented in this chapter is a result of joint work with Reza Reyhanitabar and Bart Mennink which was published in ASIACRYPT 2015 [MRV15].

Organization of the Chapter. We first discuss related work in Section 6.1 and list the contributions in Section 6.2.

We then briefly introduce the basics of the (original) Sponge and Duplex Construction in Section 6.3 and introduce the new full-state version of the keyed Sponge and keyed Duplex in Section 6.4.

In Section 6.5, we define the security model for the Full-state Keyed Sponge and Duplex, and then analyse the security of FKS in Section 6.6 and the security of FKD in Section 6.7.

Finally, we define the *Full-state SpongeWrap* (FSW) construction for AE and analyse its security in Section 6.8.

6.1 Related Work

The Sponge construction for cryptographic hashing was first introduced by Bertoni, Daemen, Peeters and Van Assche [BDPV07]. The Sponge-based hash function Keccak [BDPV08] was standardised in the SHA3 hashing standard [oST15].

Various *keyed* constructions based on the Sponge were then proposed: reseedable pseudorandom number generators [BDPA10], pseudorandom functions and message authentication codes (PRFs/MACs) [BDPV11, BDPV12], Extendable-Output Functions

("XOFs") [Per14] and authenticated encryption (AE) modes [BDPA11a, BDPV12].

The keyed Sponge construction also got adopted in Spritz, a new RC4-like stream cipher [RS14], and also in 10 out of 57 submissions to the first round of the CAESAR competition [Ber14a, AFL16]: Artemia [AAB14], Ascon [DEMS14], ICEPOLE [MGH+14], Ketje [BDP+14a], Keyak [BDP+14b], NORX [AJN14], π-Cipher [GMS+14], PRIMATEs [ABB+14a], Prøst [KLL+14] and STRIBOB [SB14]. An enhanced variant of the Full-state Keyed Duplex construction described in this chapter was adopted by the candidate Keyak [BDP+16b].

Bertoni et al. [BDPA08] proved that the keyless Sponge construction is a secure hash function up to the $O(2^{c/2})$ birthday-type bound in the indifferentiability framework of Maurer, Renner and Holenstein [MRH04].

Message Authentication. Bertoni et al. [BDPV11] introduced the keyed Sponge as a simple evaluation of the Sponge function on the concatenation of the key and the message, and proved a security bound that quantitatively improved over the indifferentiability result. Chang et al. [CDH+] considered a slight variant of the keyed Sponge where the key is processed in the inner part of the Sponge, and observed that it can be seen as the Sponge based on an Even-Mansour blockcipher [EM91, EM97]. At FSE 2015, Andreeva, Daemen, Mennink and Van Assche [ADMA15] considered a generic and improved analysis of both the outer- and inner-keyed Sponge. We reuse a part of their security analysis. So far, however, these constructions have only been considered with the classical r-bit absorption.

The idea of using *full-state* message absorption for achieving higher efficiency was first made explicit in the Donkey Sponge MAC construction [BDPV12],[1] but without any formal security proof. The recently introduced Donkey-inspired MAC function Chaskey [MMH+14] did get a formal security analysis, but its proof is specific to Chaskey and does not apply to the Donkey Sponge.

A thorough analysis of the full-state message absorption keyed Sponge was done by Gaži, Pietrzak and Tessaro [GPT15], who prove nearly tight security up to $O(\ell q(q+N)/2^b + q(q+\ell+N)/2^c)$, where the adversary makes q queries of maximal length ℓ, and makes N primitive calls. However, their analysis only applies to the *fixed-output-length* variant, and the proof does not directly extend to the original *arbitrary-output-length* keyed Sponge. In this work, we provide a direct proof for this more general case.

Authenticated Encryption. Encryption via the Sponge is typically done with the Duplex construction [BDPA11a]. Bertoni et al. showed that the Duplex allows for authenticated encryption in the form of SpongeWrap [BDPA11a]. This mode is, de facto, the basis of the majority of Sponge-based submissions to the CAESAR competition. Jovanovic et al. [JLM14] re-investigated the security of the Sponge-based authenticated encryption scheme NORX, and proved its beyond birthday-bound (in the capacity c)

[1]We note that apart from full-state absorption, the Donkey Sponge also uses less rounds in the underlying iterated permutation during the absorbing phase.

security. These results are, however, all for the usual r-bit absorption. Yasuda and Sasaki [SY15] have considered several full-state and *partially* full-state Sponge-based authenticated encryption schemes for efficient incorporation of associated data, directly lifting Jovanovic et al.'s security proofs.[2] A technique similar to full-state data absorption was also proposed by Reyhanitabar, Vaudenay and Vizár [RVV15] in their compression function based AE mode p-OMD (see Section 5.3); both p-OMD and the Full-state keyed sponge were derived from sequential algorithms that maintain secret states with AXU-like properties, which allow data to be xored into the state without harming the security.

Later, Daemen et al. [DMA17] improved the construction and security analysis of Full-state Keyed Duplex presented in this chapter.

6.2 Contribution

We present the Full-state Keyed Sponge, and Full-state Keyed Duplex, two modes of operation for a cryptographic permutation. Both of these extend the existing results on sponge-based cryptography. We analyse the security of both FKS and FKD in the ideal permutation model and show how to construct an efficient AE scheme using FKD.

FKS and FKD are the first sponge-modes that allow the most efficient kind of full-state absorption, have an arbitrary-length output, and are provably secure at the same time. Our security analysis is modular, easy to understand, and therefore easy to verify.

We demonstrate the application of FKD to AE through the Full-state SpongeWrap construction. It uses full-state absorption to process the associated data as efficiently as possible, in most cases almost for free compared to the original SpongeWrap. Moreover, most of the existing sponge-based AE schemes can easily switch to full-state absorption and benefit from our security analysis, thanks its modularity, which uses FKD as an intermediate step.

6.3 The Sponge and Duplex Constructions

The classical Sponge construction [BDPV07] takes as parameter a cryptographic permutation $p : \{0,1\}^b \to \{0,1\}^b$, an integer $0 < r < b$ and an injective padding scheme pad_r, which injectively maps any string $M \in \{0,1\}^*$ to an $\bar{M} = \mathrm{pad}_r(M)$ such that $|\bar{M}| \equiv 0 \pmod r$ and such that the last r bits of $\bar{M}$ are non-zero.[3] The instance

$$\mathrm{Sponge}[p, r, \mathrm{pad}_r] : \{0,1\}^* \times \mathbb{N}^+ \to \{0,1\}^*$$

[2]The concurrent absorption mode proposed by Yasuda and Sasaki (Fig. 3 in [SY15]) fails to utilize the full-state absorption when the associated data becomes longer than the message, forcing the mode switch from a full-state mode to the classical r-bit absorbing Sponge mode; hence, we refer to this as a *partially* full-state AE mode.

[3]The final block of r bits being non-zero is a sufficient condition on the padding scheme, to ensure the sponge indifferentiable from a random oracle. The precise condition is that for all $M \neq M'$ we must have $\mathrm{pad}_r(M) \neq \mathrm{pad}_r(M')\|0^{nr}$ for all $n \geq 0$. If this condition was violated for some M, M' then for any ℓ, $\mathrm{Sponge}[p, r, \mathrm{pad}](M, \ell)$ would be a suffix of $\mathrm{Sponge}[p, r, \mathrm{pad}](M, \ell + nr)$.

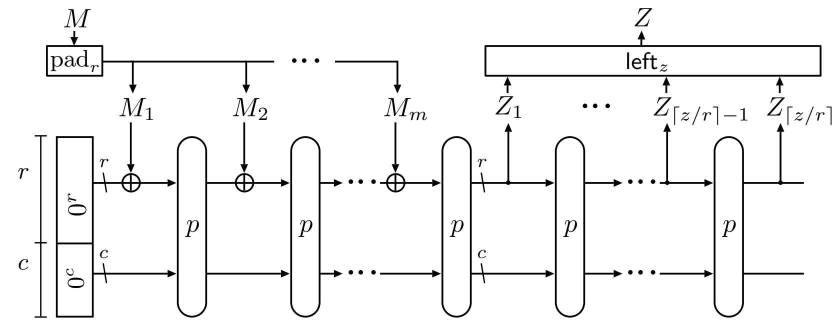


Figure 6.1 – **The computation of $Z = \text{Sponge}[p, r, \text{pad}_r](M, z)$.**

then allows to map an arbitrary input string $M \in \{0,1\}^*$ to a z-bit output string $\text{Sponge}[p, r, \text{pad}_r](M, z)$ for an arbitrary output length z.

An evaluation of the Sponge consists of a sequential application of the permutation p on a state of b bits. This state is partitioned into an r-bit outer part and a c-bit inner part, where $b = r + c$. We call r rate and c capacity of the instance. In the absorption phase, message blocks of size r bits are absorbed by the outer part and the state is transformed using p, while in the squeezing phase, digests are extracted from the outer part, r bits at a time. This is illustrated in Figure 6.1.

The Duplex construction [BDPA11a] is a stateful variant of the Sponge that allows to extract a limited number of output bits after the processing of each input block. It also takes as parameter a cryptographic permutation $p : \{0,1\}^b \to \{0,1\}^b$, an integer $0 < r < b$ and an injective padding scheme pad_r which injectively maps any string $M \in \{0,1\}^*$ to an $\bar{M} = \text{pad}_r(M)$ such that $|\bar{M}| \equiv 0 \pmod{r}$ and such that the last r bits of $\bar{M}$ are non-zero.

The instance $\text{Duplex}[p, r, \text{pad}_r]$ has two interfaces. The first interface

$$\text{Duplex}[p, r, \text{pad}_r].\text{initialize}()$$

takes no input and sets up a state of b zero bits. The interface

$$Z_i = \text{Duplex}[p, r, \text{pad}_r].\text{duplexing}(M_i, z_i)$$

then takes an input string $M_i \in \mathcal{M}_{\text{pad}_r}$ and a non-negative integer $0 \leq z_i \leq$ and outputs a string $Z_i \in \{0,1\}^{z_i}$, where $\mathcal{M}_{\text{pad}_r} = \{M \in \{0,1\}^* \mid |\text{pad}_r(M)| = r\}$ is a set of strings $M \in \{0,1\}^*$ for which we have $|\text{pad}_r(M)| = r$. A sequence of Duplex calls is illustrated in Figure 6.2.

6.4 Full-State Keyed Sponge and Full-State Keyed Duplex

In this section, we define the Full-state Keyed Sponge and the Full-State Keyed Duplex. Both constructions generalize over their original predecessors by allowing the blocks of

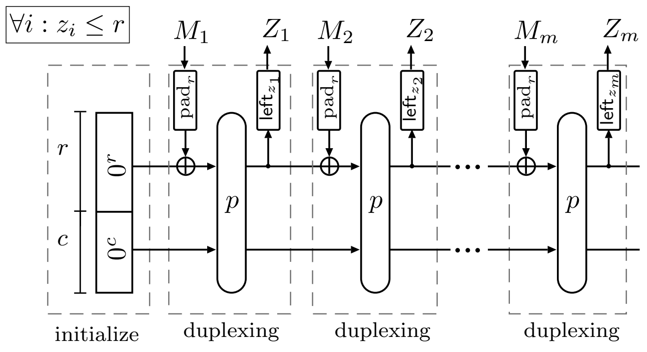


Figure 6.2 – **A sequence of calls to Duplex[$\boldsymbol{p, r, \mathrm{pad}_r}$].**

the (padded) input to have the same length as the state. This modification obviously improves the performance of both constructions, allowing more data to be processed per call to the underlying permutation.

However, the same modification also necessitates the Sponge and the Duplex to be keyed; without a key and with the control over the full state, these construction will offer no security as (keyless) hash functions. To emphasise this, we include the word *keyed* in the names of both full-state Sponge and Duplex.

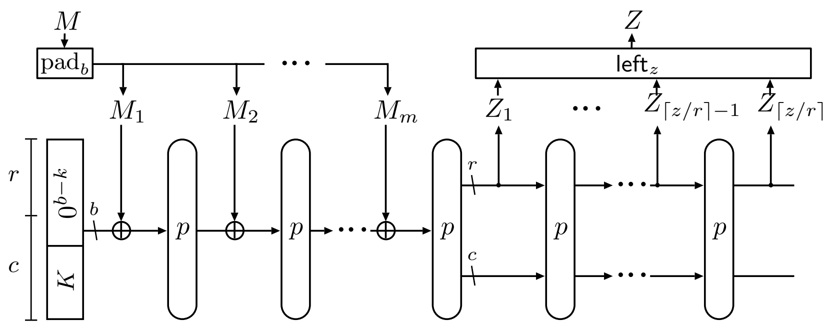


Figure 6.3 – **The FKS[$\boldsymbol{p, r, k}$] construction.**

Full-State Keyed Sponge. We define the Full-state Keyed Sponge (FKS) construction, a mode of operation for a cryptographic permutation that is parameterized by a public permutation $p : \{0,1\}^b \to \{0,1\}^b$. It is further parameterized by $r, k \in \mathbb{N}^+$, which are required to satisfy $r < b$ and $k \leq b - r$. We call k the key length, and r the *rate* of the instance and, we define $c = b - r$ and call it *capacity.* We denote by FKS$[p, r, k]$ an instance of FKS with parameters fixed to p, r and k, and we let FKS^p denote an instance of FKS using a permutation p; some or even all parameters are sometimes left implicit if they are clear from the context.

An instance

$$\mathrm{FKS}[p, r, k] : \{0,1\}^k \times \{0,1\}^* \times \mathbb{N}^+ \to \{0,1\}^*$$

is a generalization of a keyed function that allows to choose the length of the output. It takes a key $K \in \{0,1\}^k$, a message $M \in \{0,1\}^*$, and a natural number z, and it outputs a string $Z \in \{0,1\}^z$:

$$\mathrm{FKS}[p,r,k](K,M,z) = \mathrm{FKS}_K^p(M,z) = Z\,.$$

It operates on a state $t \in \{0,1\}^b$, which is initialized using the key K. The message M is first padded to a length multiple of b bits, using a padding scheme pad_b defined by $\mathrm{pad}_b(M) = M\|10^{b-1-|M| \bmod b}$. The padded input is then partitioned to m message blocks $M_1\|...\|M_m$ of b bits each. We stress that we make use of the explicitly defined padding scheme for FKS.[4] These message blocks are processed one-by-one, interleaved with evaluations of p. After the absorption of M, the outer r bits of the state are output and the state is processed via p until a sufficient number of output bits are obtained. FKS is depicted in Figure 6.3, and Figure 6.5 provides a formal specification of FKS.

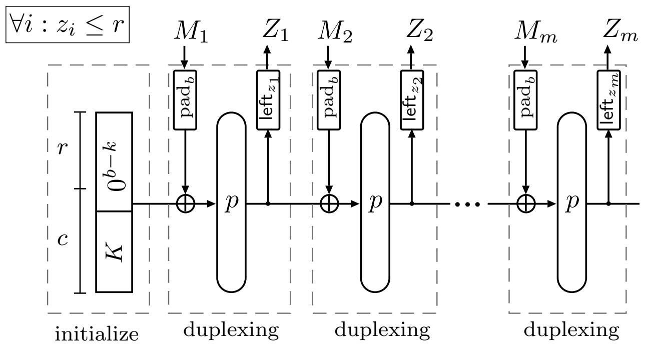


Figure 6.4 – **The FKD$[p, r, k]$ construction.**

Full-State Keyed Duplex. We define the Full-state Keyed Duplex (FKD) construction, a generalization of the Duplex of Bertoni et al. [BDPA11a, BDPA11b]. FKD is a deterministic stateful variant of the FKS that allows to extract a limited number of output bits after the processing of each input block.

It is parameterized by a public permutation $p : \{0,1\}^b \to \{0,1\}^b$, the rate $r \in \mathbb{N}^+$ and the key length $k \in \mathbb{N}^+$, which are required to satisfy $r < b$ and $k \le b - r$. We call $c = b - r$ the capacity. We let FKD$[p, r, k]$ denote an instance of FKD with parameters fixed to p, r, k, and we let FKD^p denote an instance that uses a permutation p. Again, the parametrization can be left implicit if clear from the context.

Let D denote FKD$[p, r, k]$. D has two interfaces: D.initialize and D.duplexing. D.initialize is used to set up the state of D, it gets as input a key $K \in \{0,1\}^k$ and outputs nothing. D.duplexing is used to process one input block and compute output bits, it gets as input a message block $M_i \in \{M \in \{0,1\}^* \mid |\mathrm{pad}_b(M)| = b\}$ and a natural number $0 \le z_i \le r$, and it outputs a string $Z_i \in \{0,1\}^{z_i}$. Internally, FKD uses the same

[4]While we could use any padding scheme such that $|\mathrm{pad}_b(M)| \equiv 0 \pmod b$ with the last b bits of $\mathrm{pad}_b(M)$ non-zero for all $M \in \{0,1\}^*$, we fix $\mathrm{pad}_b(M) = M\|10^{b-1-|M| \bmod b}$ for the sake of concreteness.

padding scheme as FKS. FKD is illustrated in Figure 6.4, and the formal specification is given in Figure 6.5.

```
algorithm FKS$[p, r, k](K, M, z)$
    $t \leftarrow 0^{b-k} \| K$
    $M_1 \| \cdots \| M_m \overset{b}{\leftarrow} \mathrm{pad}_b(M)$
    for $i \leftarrow 1, \ldots, m$ do
        $s \leftarrow t \oplus M_i$
        $t \leftarrow p(s)$
    end for
    $Z \leftarrow \mathsf{left}_r(t)$
    while $|Z| < z$ do
        $t \leftarrow p(t)$
        $Z \leftarrow Z \| \mathsf{left}_r(t)$
    end while
    return $\mathsf{left}_z(Z)$
end algorithm
```

```
algorithm FKD$[p, r, k]$
    interface FKD.initialize$(K)$
        $t \leftarrow 0^{b-k} \| K$
    end interface

    interface FKD.duplexing$(M, z)$
        if $z > r$ or $|\mathrm{pad}_b(M)| \neq b$ then
            return $\perp$
        end if
        $s \leftarrow t \oplus \mathrm{pad}_b(M)$
        $t \leftarrow p(s)$
        return $\mathsf{left}_z(t)$
    end interface
end algorithm
```

Figure 6.5 – **Definition of FKS$[\boldsymbol{p}, \boldsymbol{r}, \boldsymbol{k}]$ and FKD$[\boldsymbol{p}, \boldsymbol{r}, \boldsymbol{k}]$**, both parameterized with a cryptographic permutation $p : \{0,1\}^b \to \{0,1\}^b$, a rate r and a keylength k.

Notation. We introduce additional notation for this chapter. Given implicit parameters $p : \{0,1\}^b \to \{0,1\}^b, r, k$ of FKS (or of FKD) and a string $s \in \{0,1\}^b$, we let $\mathsf{outer}(s) = \mathsf{left}_r(s)$ denote the "outer" part of the string (used to produce output bits in) and $\mathsf{inner}(s) = \mathsf{right}_c(s)$ denote the "inner" part of the string (which is never revealed). Note that we have $s = \mathsf{outer}(s) \| \mathsf{inner}(s)$.

6.5 Security Model

Because the FKS is a generalization of a keyed function, we need to modify the security model to reflect the changes in syntax and functionality. The same applies to FKD, which deviates from a keyed function even further.

In addition, the underlying primitive, a cryptographic permutation, is keyless. This makes a standard model-reduction to its security difficult, because there is no compact standard-model security definition for keyless permutation.[5]

We therefore cast our security notions and analysis in the *ideal permutation model*, where the underlying cryptographic permutation used by either FKS or FKD is modelled as a *public* random permutation $p \leftarrow\$ \mathrm{Perm}(2^b)$. That is, in addition to the usual oracles, the adversary has direct oracle access to both p and its inverse p^{-1}. The parameterized

[5]Some early results in this direction appeared after the present results were published [ST17].

resources of the adversary are extended to include the total number of calls to either p or p^{-1}.

The ideal permutation model can be thought of as abstracting away any structural properties that an actual cryptographic permutation may have. A proof in the model can then be seen as a sanity check of the *construction*, which shows that the construction in question resists to generic attacks. Loosely speaking, an actual instance should be secure, as long as the used permutation does not have any "serious structural weakness". This statement is however very informal, and there is no rigorous link between the result in the idealized model and the security of actual instances.

Note that because the underlying primitive is modelled as a random permutation, we do not rely on computational security in any part of the analysis. We can therefore work with information-theoretic adversaries throughout the whole analysis.

Multiplicity. Following Andreeva et al. [ADMA15], we add the so called *multiplicity* to the parameterized resources of an adversary in the context of the ideal permutation model.

With an implicit integer b, for a set $\mathcal{S} = \{(x_i, y_i) \in (\{0,1\}^b)^2\}_{i=1}^{\sigma}$ of σ pairs (x_i, y_i), and for a rate $0 < r < b$, we define the total maximal multiplicity of S as a sum $\mu = \mu_{\text{fwd}} + \mu_{\text{bwd}}$ of the forward multiplicity μ_{fwd} and the backward multiplicity μ_{bwd}, where

$$\begin{aligned}
\mu_{\text{fwd}} &= \max_a |\{i \in \{1,\dots,\sigma\} \ : \ \mathsf{outer}(x_i) = a\}| \text{ and} \\
\mu_{\text{bwd}} &= \max_a |\{i \in \{1,\dots,\sigma\} \ : \ \mathsf{outer}(y_i) = a\}|
\end{aligned}$$

(see Section 6.4 for the definition of $\mathsf{outer}(\cdot)$). In our analysis, the set S will always be be constructed as a collection of input-output pairs of a permutation $p : \{0,1\}^b \to \{0,1\}^b$, so the multiplicity is the maximal number of permutation inputs in the collection S who share the same value in their leftmost r bits, plus the same for the permutation outputs in S.

The multiplicity is a quantity that characterises the data that are available to the adversary during an attack. We have $2 \le \mu \le 2\sigma$ per definition (the maximum $\mu = 2\sigma$ is reached e.g. when p is the identity permutation), however the upper bound 2σ is never reached in practical applications of sponge-based constructions. We note that we are always interested in the multiplicity of input-output pairs that were *induced* by adversarial queries to a *construction*, not directly made by the adversary.

Being a sum of forward and backward multiplicities, the total multiplicity can be seen as a measure of adversary's ability to control the outer part of the permutation inputs and outputs respectively. In case of sponge-based designs, the backward multiplicity can be expected to be approximately $\sigma 2^{-r}$ if σ blocks of data are processed with a single key while the forward multiplicity varies with concrete applications [ADMA15].

6.5.1 Security Model for FKS

We first define *keyed functions with extendible output*, the kind of object that FKS actually is. A keyed function with extendible output is an efficient algorithm $F : \mathcal{K} \times \mathcal{D} \times \mathbb{N}^+ \to \{0,1\}^*$ that maps a secret key K, an input M, and a desired output length z to an output string $F(K, M, z) \in \{0,1\}^z$. We call $\mathcal{K}$, a finite set, the key space of F. We let $F_K(M, z) = F(K, M, z)$. If F is a mode of operation for a low-level primitive p, we let F^p denote an instance of F using p.

An instance $F = \text{FKS}[p, r, k]$ with permutation $p : \{0,1\}^b \to \{0,1\}^b$, rate r and key length k is thus a keyed function with extendible output having $\mathcal{K} = \{0,1\}^k$ and $\mathcal{D} = \{0,1\}^*$.

Upon inspection of Figure 6.5, we see that when fed with (K, M, z_1) and (K, M, z_2) with arbitrary $1 \leq z_1 < z_2$, $K \in \{0,1\}^k$ and $M \in \{0,1\}^*$, the FKS will always produce outputs Z_1 and Z_2 such that Z_1 is a prefix of Z_2. This is because the output is always derived from the state right after the processing of the message, which does not depend on z_i.

The PRF security of a keyed function with extendible output is defined with help of games **prfx-R** and **prfx-I** in Figure 6.6. We make sure that the idealized reference object for the FKS in the game **prfx-I** in Figure 6.6 does have this inherent property of the construction, but is random otherwise. For each input (M, z), we first check if any output bits were generated, and whether their length is smaller than z. If any of the two occurs, we generate the required number of uniform bits. This way, we always satisfy the prefix property.

Definition 6.1 (PRF-x security). *Given a keyed function with extendible output F which is a mode of operation for a b-bit cryptographic permutation, and an information-theoretic adversary $\mathscr{A}$ that has black-box access to F, we define the advantage of $\mathscr{A}$ in breaking the security of F as*

$$\mathbf{Adv}_F^{\mathbf{prfx}}(\mathscr{A}) = \Pr\left[\mathscr{A}^{\mathbf{prfx}\text{-}\boldsymbol{R}_F} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathbf{prfx}\text{-}\boldsymbol{I}_F} \Rightarrow 1\right].$$

If the advantage $\mathbf{Adv}_F^{\mathbf{prfx}}(\mathscr{A}) \leq \epsilon$ for every adversary $\mathscr{A}$ that makes no more than N queries to p and p^{-1} in total, that makes no more than q Eval *queries that have the total maximal multiplicity limited by μ such that each* Eval *query induces no more than ℓ calls to the underlying permutation, we say that F is a $(\epsilon, q, \ell, \mu, N)$-secure* pseudorandom function with extendible output (PRF-x).

6.5.2 Security Model for FKD

We now define *duplexing keyed functions*, which are the kind of object that FKD is. A duplexing keyed function with a key space $\mathcal{K}$, rate r and input limit b is an efficient, stateful algorithm F that exposes two interfaces. The interface F.initialize(K) initializes F with a secret key $K \in \mathcal{K}$ and returns no output. The interface $Z = F$.duplexing(M, z)

```
proc initialize                                   $\boxed{\textbf{prfx-R}_F}$
$\diamond$ $\pi \leftarrow\$ \mathrm{Perm}(2^b)$
$p \leftarrow\$ \mathrm{Perm}(2^b)$
$K \leftarrow\$ \mathcal{K}$

proc $p(x), p^{-1}(y)$

proc Eval$(M, z)$
$\diamond$ return $F^{\pi}(0^k, M, Z)$
return $F^{p}(K, M, Z)$
```

```
proc initialize                                   $\boxed{\textbf{prfx-I}_F}$
$p \leftarrow\$ \mathrm{Perm}(2^b)$
$\mathcal{X} \leftarrow \mathrm{array}(\{0,1\}^*)$

proc $p(x), p^{-1}(y)$

proc Eval$(M, z)$
if $\mathcal{X}[M] = \perp$ then
  $\mathcal{X}[M] \leftarrow\$ \{0,1\}^z$
elsif $z > |\mathcal{X}[M]|$ then
  $Z \leftarrow\$ \{0,1\}^{z-|\mathcal{X}[M]|}$
  $\mathcal{X}[M] \leftarrow \mathcal{X}[M] \| Z$
return $\mathsf{left}_z\,(\mathcal{X}[M])$
```

```
proc initialize                                   $\boxed{\textbf{prfd-R}_F}$
$\diamond$ $\pi \leftarrow\$ \mathrm{Perm}(2^b)$
$p \leftarrow\$ \mathrm{Perm}(2^b)$
$K \leftarrow\$ \mathcal{K}$
$\mathsf{init} \leftarrow \mathsf{false}$

proc $p(x), p^{-1}(y)$

proc Initialize()
$\mathsf{init} \leftarrow \mathsf{true}$
$F^{p}$.initialize$(K)$
$\diamond$ $F^{\pi}$.initialize$(0^k)$
return $\perp$

proc Duplexing$(M, z)$
if $\mathsf{init} = \mathsf{false}$ or $z > r$ or $|\mathrm{pad}_b M| \neq b$ then
  return $\perp$
$\diamond$ return $F^{\pi}$.duplexing$(M, z)$
return $F^{p}$.duplexing$(M, z)$
```

```
proc initialize                                   $\boxed{\textbf{prfd-I}_F}$
$p \leftarrow\$ \mathrm{Perm}(2^b)$
$\mathcal{X} \leftarrow \mathrm{array}((\{0,1\}^*)^*)$
$\mathbf{t} \leftarrow \perp$

proc $p(x), p^{-1}(y)$

proc Initialize()
$\mathbf{t} \leftarrow \Lambda$
return $\perp$

proc Duplexing$(M, z)$
if $\mathbf{t} = \perp$ or $z > r$ or $|\mathrm{pad}_b M| \neq b$ then
  return $\perp$
$\mathbf{t} \leftarrow \mathbf{t} \| M$
if $\mathcal{X}[t] = \perp$
  $\mathcal{X}[t] \leftarrow\$ \{0,1\}^r$
return $\mathsf{left}_z\,(\mathcal{X}[\mathbf{t}])$
```

Figure 6.6 – **Games for defining security of keyed functions with extendible output length (PRF-x security) and security of duplexing keyed functions (PRF-d).** The games **prfx-R** and **prfd-R** *do not* include the lines marked with $\diamond$ (the full codes, including the lines marked with $\diamond$ respectively define the games $\diamond$**prfx-R** and $\diamond$**prfd-R**). Note that the permutation p is exposed to the adversary. Note that in the **prfd-I**$_F$ game, the variable **t** is a *list* of strings (see Section 2.1 for definition of a list.)

takes an input string $M \in \{0,1\}^{<b}$ and the desired output length $z \le r$, and returns the output value $Z \in \{0,1\}^z$. If F is a mode of operation for a low-level primitive p, we let F^p denote an instance of F using p.

An instance $F = \mathrm{FKD}[p,r,k]$ with permutation $p : \{0,1\}^b \to \{0,1\}^b$, rate r and key length k is thus a duplexing keyed function that has a rate r, input limit b and $\mathcal{K} = \{0,1\}^k$.

It can be seen from Figure 6.5 that if we initialize the FKD twice with the same key and then duplex the same sequence of input blocks $M_1, \ldots, M_\ell$, with two sequences of output lengths $z_1^1, \ldots, z_\ell^1$ and $z_1^2, \ldots, z_\ell^2$ such that $z_i^1 < z_i^2$ for all $1 \le i \le \ell$, then an output block Z_i^1 will always be a prefix of the corresponding block Z_i^2 for all $1 \le i \le \ell$. Intuitively, we expect that if we then process $M_{\ell+1}^1 \ne M_{\ell+1}^2$, the corresponding output blocks $Z_{\ell+1}^1$ and $Z_{\ell+1}^2$ should be independent.

The PRF security of a duplexing keyed function is defined with help of games **prfd-R** and **prfd-I** in Figure 6.6. The idealized reference object in the game **prfd-I** in Figure 6.6 formalizes the intuition we just described. The game keeps a list **t** of all input blocks queried from the last Initialize query, and keeps an array $\mathcal{X}$ that stores r random bits for every unique value of **t** that occurred in the game.

Definition 6.2 (PRF-d security)**.** *Given a duplexing keyed function F with key space $\mathcal{K}$, rate r and input limit b, which is a mode of operation for a b-bit cryptographic permutation, and given an information-theoretic adversary $\mathscr{A}$ that has black-box access to F, we define the advantage of $\mathscr{A}$ in breaking the security of F as*

$$\mathbf{Adv}_F^{\mathbf{prfd}}(\mathscr{A}) = \Pr\left[\mathscr{A}^{\mathbf{prfd}\text{-}\boldsymbol{R}_F} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathbf{prfd}\text{-}\boldsymbol{I}_F} \Rightarrow 1\right].$$

If the advantage $\mathbf{Adv}_F^{\mathbf{prfd}}(\mathscr{A}) \le \epsilon$ for every adversary $\mathscr{A}$ that makes no more than N queries to p and p^{-1} in total, that makes no more than q Initialize *queries, such that each such query is followed by no more than ℓ* Duplexing *queries with the total maximal multiplicity limited by μ, we say that F is a $(\epsilon, q, \ell, \mu, N)$-secure* duplexing pseudorandom function (PRF-d).

6.5.3 Security Model for Even-Mansour

Our proof relies on a reduction to the security of a low-entropy, single-key Even-Mansour construction [EM91, EM97] that turns a b-bit cryptographic permutation into a b-bit blockcipher with a key of $k < b$ bits. In more detail, let $p : \{0,1\}^b \to \{0,1\}^b$ be a permutation, $k < b$ be the key length and $K \in \{0,1\}^k$ be a key. The Even-Mansour blockcipher $E^p \colon \{0,1\}^k \times \{0,1\}^b \to \{0,1\}^b$ is defined as

$$E_K^p(M) = p(M \oplus (0^{b-k} \| K)) \oplus (0^{b-k} \| K).$$

To analyse the generic PRP security of E^p, we need to re-cast the Definition 2.3 in the ideal permutation model. In this chapter, we overload the definition of the PRP adversarial advantage to

$$\mathbf{Adv}^{\mathbf{prp}}_{E^p}(\mathscr{A}) = \Pr\left[K \leftarrow_\$ \{0,1\}^k, p \leftarrow_\$ \mathrm{Perm}(2^b) \ : \ \mathscr{A}^{E^p_K, p, p^{-1}} \Rightarrow 1\right] - \Pr\left[\pi, p \leftarrow_\$ \mathrm{Perm}(2^b) \ : \ \mathscr{A}^{\pi, p, p^{-1}} \Rightarrow 1\right].$$

We remove time t, and add the number of p and p^{-1} queries and the total maximal multiplicity μ of all construction queries to the parameterized adversarial resources.

6.6 Security of Full-State Keyed Sponge

We prove the following result for FKS:

Theorem 6.3. *Let $b, r, c, k > 0$ be such that $b = r + c$ and $k \leq c$. Let* FKS *be the scheme defined in Figure 6.5. Let $\mathscr{A}$ be an adversary that makes no more than N queries to p and p^{-1} in total, that makes no more than q* Eval *queries that have the total maximal multiplicity limited by μ such that each* Eval *query induces no more than ℓ calls to the underlying permutation. Then,*

$$\mathbf{Adv}^{\mathbf{prfx}}_{\mathrm{FKS}}(\mathscr{A}) \leq \frac{2(q\ell)^2}{2^b} + \frac{2q^2\ell}{2^c} + \frac{\mu N}{2^k}.$$

The proof of Theorem 6.3 follows to a certain extent the modular approach of Andreeva et al. [ADMA15]. In particular we use the fact that the instance FKS^p_K can alternatively be viewed as $\mathrm{FKS}^{E^p_K}_0$, where the underlying permutation is replaced by the low-entropy Even-Mansour blockcipher that also absorbs the key (see Figure 6.7). This clever observation was used before by Chang et al. [CDH$^+$]. Note that this observation only works for $k \leq c$: it relies on xoring two dummy keys $K \oplus K$ in-between every two adjacent permutation calls, and if $k > c$, there would be $k - c$ bits of the FKS-state that the adversary would see unkeyed.

This trick allows to split the security analysis of FKS^p_K into two steps; we first do the security analysis of the Even-Mansour blockcipher and then the security analysis of FKS instantiated with a *secret* permutation. This is where we diverge from the approach of Andreeva et al. [ADMA15], who simply applied the classical indifferentiability result of [BDPA08] to deal with the security analysis of the Keyed Sponge instantiated with a secret permutation. Because the indifferentiability bound cannot be used for FKS due to its full-state absorption, we carry out a new analysis and derive an improved bound. In addition, applying the indifferentiability result gives a rather loose bound which we improve with our new analysis.

Throughout the analysis, we will assume, without loss of generality, that the adversary $\mathscr{A}$ makes exactly q queries, such that each query induces exactly ℓ calls to the underlying permutation. We further let $\Delta_{\mathscr{A}}(G_1; G_2)$ denote the expression

$$\Pr\left[\mathscr{A}^{G_1} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{G_2} \Rightarrow 1\right]$$

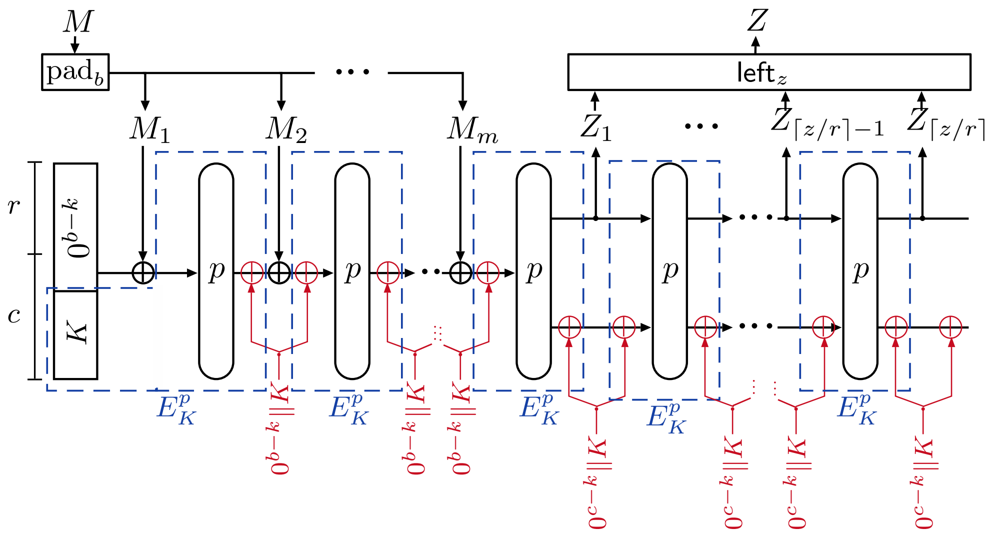


Figure 6.7 – **The equivalence between $\mathbf{FKS}_K^p$ and $\mathbf{FKS}_0^{E_K^p}$.** Xoring the two dummy keys to the state between each two p-calls (in red) does not change the value of Z. Two keys around each p-call are then absorbed by a single E_K^p evaluation (blue boundary). Note that since the inner c bits of the state are never revealed, the final copy of the key that is "left" does not invalidate this claim.

for two games G_1 and G_2.

Proof of Theorem 6.3. We define the game $\diamond$**prfx-R** by including the lines marked by $\diamond$ in the game **prfx-R** in Figure 6.6. By doing so, we replace the public permutation p by an independent secret permutation π in all the primitive calls made by the FKS. We use this game as an intermediate step in the analysis, and define

$$\mathbf{Adv}_{\mathrm{FKS}}^{\diamond\mathbf{prfx}}(\mathscr{A}) = \Pr\left[\mathscr{A}^{\diamond\mathbf{prfx\text{-}R}_{\mathrm{FKS}}} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathbf{prfx\text{-}I}_{\mathrm{FKS}}} \Rightarrow 1\right]$$

for the sake of tidiness of the proof.

We have that $\mathrm{FKS}_K^p = \mathrm{FKS}_0^{E_K^p}$ (see Figure 6.7). Abusing the Δ-notation for the PRP security games, it follows that

$$\begin{aligned}
\mathbf{Adv}_{\mathrm{FKS}}^{\mathbf{prfx}}(\mathscr{A}) &= \Delta_{\mathscr{A}}\left(\mathbf{prfx\text{-}R}_{\mathrm{FKS}}; \mathbf{prfx\text{-}I}_{\mathrm{FKS}}\right) \\
&= \Delta_{\mathscr{A}}\left(\mathbf{prfx\text{-}R}_{\mathrm{FKS}}; \diamond\mathbf{prfx\text{-}R}_{\mathrm{FKS}}\right) + \Delta_{\mathscr{A}}\left(\diamond\mathbf{prfx\text{-}R}_{\mathrm{FKS}}; \mathbf{prfx\text{-}I}_{\mathrm{FKS}}\right) \\
&\leq \Delta_{\mathscr{B}}\left(p, E_K^p; p, \pi\right) + \Delta_{\mathscr{C}}\left(\diamond\mathbf{prfx\text{-}R}_{\mathrm{FKS}}; \mathbf{prfx\text{-}I}_{\mathrm{FKS}}\right) \\
&\leq \mathbf{Adv}_{E^p}^{\mathbf{prp}}(\mathscr{B}) + \mathbf{Adv}_{\mathrm{FKS}}^{\diamond\mathbf{prfx}}(\mathscr{C})
\end{aligned}$$

for some adversary $\mathscr{B}$ that makes $q \cdot \ell$ construction queries that have total maximal multiplicity μ and makes N queries to p and p^{-1}, and some adversary $\mathscr{C}$ that makes q queries such that each query induces at most ℓ calls to the underlying permutation.

This is because $\mathscr{B}$ uses its own construction oracle to simulate FKS for $\mathscr{A}$, forwards

$\mathscr{A}$'s p and p^{-1} queries to its own oracles, and in the end outputs whatever $\mathscr{A}$ outputs; $\mathscr{B}$ then perfectly simulates either $\mathbf{prfx\text{-}R}_{\mathrm{FKS}}$ or $\diamond\mathbf{prfx\text{-}R}_{\mathrm{FKS}}$ for $\mathscr{A}$.

$\mathscr{C}$ only forwards $\mathscr{A}$'s queries to its own corresponding oracles and forwards $\mathscr{A}$'s final output; the simulation for $\mathscr{A}$ is trivially perfect.

Note that $\mathscr{C}$ also has access to p and p^{-1}, but queries to this oracle are useless as its Eval oracle is independent of p in both games.

Andreeva et al. proved that $\mathbf{Adv}^{\mathbf{prp}}_{E^p}(\mathscr{B}) \leq \frac{\mu N}{2^k}$ for any $\mathscr{B}$ [ADMA15]. In Lemma 6.4, we prove that $\mathbf{Adv}^{\diamond\mathbf{prfx}}_{\mathrm{FKS}}(\mathscr{C}) \leq \frac{2(q\ell)^2}{2^b} + \frac{2q^2\ell}{2^c}$ for any adversary $\mathscr{C}$. □

Lemma 6.4. *Let $b, r, c > 0$ be such that $b = r + c$. Let $\mathscr{A}$ be an adversary that makes no more than q* Eval *queries such that each* Eval *query induces no more than ℓ calls to the underlying permutation. Let* FKS *be the scheme of defined in Figure 6.5. Then,*

$$\mathbf{Adv}^{\diamond\mathbf{prfx}}_{\mathrm{FKS}}(\mathscr{A}) \leq \frac{2(q\ell)^2}{2^b} + \frac{2q^2\ell}{2^c}.$$

Proof. We denote the queries made by $\mathscr{A}$ as $(M^1, \zeta_1), \dots, (M^q, \zeta_q)$ and the corresponding outputs of the FKS as $Z^1, \dots, Z^q$ (we switch to ζ_i because we will use z_i to denote the number of blocks in Z^i). We denote the blocks of the i^{th} input message by $M^i_1 \| \dots \| M^i_{m_i} \xleftarrow{b} M^i$ with $m_i = |M^i|_b$. We denote the blocks of the i^{th} output value by $Z^i_1 \| \dots \| Z^i_{z_i} \xleftarrow{b} Z^i$ with $z_i = |Z^i|_r$.

Given that the 10^* padding is applied to every query, publicly known and injective, we can simplify the analysis and assume that all the queries are already padded. I.e. we assume that for $1 \leq i \leq q$, the query M^i has length divisible by b and that $M^i_{m_i} \neq 0^b$.

We further assume, that the adversary always asks for output of length divisible by r, i.e. $\zeta_i \equiv 0 \pmod r$ for all $1 \leq i \leq q$, and that every query induces *exactly* ℓ primitive calls. This is without loss of generality: we can simply give "free bits" to the adversary upon every query without decreasing its advantage.

We will denote the b-bit state of FKS just *before* the j^{th} application of π is made when processing the i^{th} query as s^i_j for $1 \leq j \leq \ell$. Similarly, we will denote the b-bit state of FKS just *after* the j^{th} application of π in i^{th} query as t^i_j for $1 \leq j \leq \ell$. We will call the former in-states and the latter out-states. Note that every in-state s^i_j is determined by the out-state t^i_{j-1} and the message block M^i_j as $s^i_j = t^i_{j-1} \oplus M^i_j$ in the absorbing phase or just by t^i_j in the squeezing phase as depicted in Fig. 6.8.
To aid the simplicity of further analysis, we additionally define initial dummy out-states $t^i_0 = 0^b$ and extended queries $\bar{M}^i = M^i \| 0^{(\ell - m_i)b}$ for $1 \leq i \leq q$. Now we can express every in-state, be it absorbing or squeezing, as $s^i_j = t^i_{j-1} \oplus \bar{M}^i_j$. We will group the out-states of i^{th} query as $T^i = (t^i_0, t^i_1, \dots, t^i_\ell)$.

Because each query induces exactly ℓ calls to π, we know that a query M^i will be answered by a string $Z_i = Z^i_1 \| \dots \| Z^i_{z_i}$ with $z_i = \ell - m_i + 1$ and $|Z^i_j| = r$ for $1 \leq j \leq z_i$. In particular, we have that $Z^i_j = \mathsf{outer}\left(t^i_{m_i+j-1}\right)$.

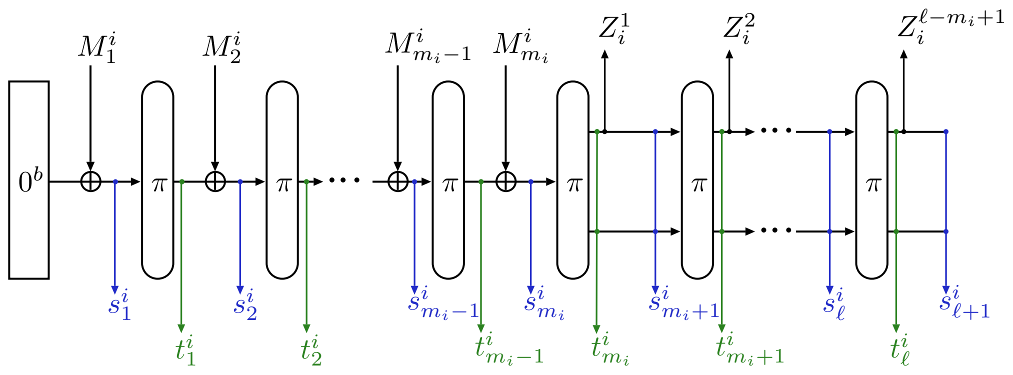


Figure 6.8 – **Internal states of the FKS.** When processing a query M^i (using a secret permutation π), the internal state of the FKS takes all of the depicted values.

The RP-RF Switch We define the game $\diamond$**prfx-R**^{f} by replacing the random permutation $\pi \leftarrow\$ \text{Perm}(2^b)$ by a random function $f \leftarrow\$ \text{Func}(2^b)$ in the game $\diamond$**prfx-R**. We have

$$\begin{aligned}\mathbf{Adv}^{\diamond\mathbf{prfx\text{-}R}}_{\text{FKS}}(\mathscr{A}) &= \Delta_{\mathscr{A}}\left(\diamond\mathbf{prfx\text{-}R}_{\text{FKS}};\diamond\mathbf{prfx\text{-}R}^f_{\text{FKS}}\right) + \Delta_{\mathscr{A}}\left(\diamond\mathbf{prfx\text{-}R}^f_{\text{FKS}};\mathbf{prfx\text{-}I}_{\text{FKS}}\right)\\ &\le (q\ell)^2/2^b + \Delta_{\mathscr{A}}\left(\diamond\mathbf{prfx\text{-}R}^f_{\text{FKS}};\mathbf{prfx\text{-}I}_{\text{FKS}}\right).\end{aligned}$$

The proof of this claim is a simple hybrid argument, where an RP-RF distinguisher $\mathscr{B}$ simulates either $\diamond$**prfx-R**$_{\text{FKS}}$ or $\diamond$**prfx-R**$^f_{\text{FKS}}$ for $\mathscr{A}$ using its own oracle instead of the permutation π. The distinguishing advantage of $\mathscr{B}$ is limited by $(q\ell)^2/2^b$ due to Lemma 2.6.

Patarin's Coefficient-H Technique We use the coefficient-H technique (see Section 2.2) to show that $\Delta_{\mathscr{A}}\left(\diamond\mathbf{prfx\text{-}R}^f_{\text{FKS}};\mathbf{prfx\text{-}I}_{\text{FKS}}\right) \le (q\ell)^2/2^b + 2q^2\ell/2^c$. The two games the adversary is trying to distinguish are $\diamond$**prfx-R**$^f_{\text{FKS}}$ and **prfx-I**$_{\text{FKS}}$. We will refer to the former as interchangeably as the "real world" or simply as X, and to the latter interchangeably as the "ideal world" or simply as Y. In either of the games, the adversary makes q queries $M^1, \ldots, M^q$ and learns the responses $Z^1, \ldots, Z^q$. The transition from queries M^i to $\bar{M}^i$ is injective, and additionally the length M^i of M^i is implicit from $\bar{M}^i$. Therefore, we can summarize the interaction of the adversary with its oracle (X or Y) with a transcript $(\bar{M}^1, \ldots, \bar{M}^q, Z_1, \ldots, Z_q)$.

To facilitate the analysis, we will disclose additional information $T^1, \ldots, T^q$ to the adversary *at the end of the experiment*. In the real world, these are the out-states $T^i = (t_0^i, t_1^i, \ldots, t_\ell^i)$ for $1 \le i \le q$, as defined at the beginning of the proof.

In the ideal world, these are dummy variables that we generate at the end of the experiment. For $1 \le i \le q$ and $0 \le j \le \ell$, we set $t_j^i \leftarrow\$ \{0,1\}^b$ independently, except for these cases:

1. $t_0^i \leftarrow 0^b$ for $1 \le i \le q$,

2. if $\mathsf{llcp}_b\left(\bar{M}^i, \bar{M}^{i'}\right) = n$ for $1 \le i' < i \le q$ then $t_j^i \leftarrow t_j^{i'}$ for $1 \le j \le n$,

3. force $\mathsf{outer}\left(t_{j+m_i-1}^i\right) = Z_j^i$ for $1 \le i \le q$ and $1 \le j \le z_i$.

Note that in both worlds, $Z^1, \ldots, Z^q$ are fully determined by $T^1, \ldots, T^q$, so we can drop them from the transcript. Thus a transcript of adversary's interaction with FKS will be $\tau = (\bar{M}^1, \ldots, \bar{M}^q, T^1, \ldots, T^q)$.

Recall that to use the Coefficient-H Technique, we define the two distributions of transcripts $D_X^{\mathscr{A}}$ and $D_Y^{\mathscr{A}}$ induced by $\mathscr{A}$ and the games X and Y. We further consider the set of attainable transcripts $\mathcal{T} = \{\tau | \Pr[D_Y^{\mathscr{A}}] > 0\}$. Referring to Lemma 2.2, we will show that there exists a set of bad transcripts $\mathcal{T}_{\text{bad}} \subset \mathcal{T}$, such that

$$\Pr\left[D_X^{\mathscr{A}} = \tau\right] / \Pr\left[D_Y^{\mathscr{A}} = \tau\right] = 1$$

for any $\tau \in \mathcal{T}_{\text{good}} = \mathcal{T} \backslash \mathcal{T}_{\text{bad}}$, and thus $\Delta_{\mathscr{A}}\left(\diamond\mathbf{prfx}\text{-}\mathbf{R}_{\text{FKS}}^f; \mathbf{prfx}\text{-}\mathbf{I}_{\text{FKS}}\right) \le \Pr\left[D_Y^{\mathscr{A}} \in \mathcal{T}_{\text{bad}}\right]$.

Definition of a Bad Transcript We label a transcript τ as *bad* if

$$\begin{aligned}&\exists (1,1) \le (i', j') \neq (i, j) \le (q, \ell) \text{ such that:}\\ &\quad\left((j \neq j') \vee \left(\mathsf{llcp}_b\left(\bar{M}^i, \bar{M}^{i'}\right) < j = j' \le \ell\right)\right) \wedge \left(t_{j-1}^i \oplus \bar{M}_j^i = t_{j'-1}^{i'} \oplus \bar{M}_{j'}^{i'}\right).\end{aligned} \tag{6.1}$$

The set of bad transcript is then $\mathcal{T}_{\text{bad}} = \{\tau \in \mathcal{T} | \tau \text{ is } \textit{bad}\}$. This definition of a bad transcript comes with an intuitive, informal interpretation; as long as all relevant *inputs* $s_j^i = t_{j-1}^i \oplus \bar{M}_j^i$ to the random function f induced by the Sponge function are distinct the output of the Sponge will be distributed uniformly. We do not require uniqueness of all in-states because the adversary can trivially force their repetition by issuing queries with common prefixes. However these collisions are not a problem, because uniqueness of the queries implies that $\mathsf{llcp}_b\left(\bar{M}^i, \bar{M}^{i'}\right) < \max\{m_i, m_{i'}\}$ for any two queries $\bar{M}^i, \bar{M}^{i'}$. Even if the adversary truncates an old query and thus forces an old absorbing in-state s to be squeezed for output, it is still not a problem because the adversary *has not seen* the image $f(s)$ before. We note that even though in-states do not exist in the ideal world, they can be defined by the same relation as in the real world, i.e. $s_j^i = t_{j-1}^i \oplus \bar{M}_j^i$.

Bounding the Ratio of Probabilities of Good Transcripts In the ideal world, the out-states $t_0^i = 0^b$ for $1 \le i \le q$ are always assigned their value trivially. We also trivially assign a single uniform b-bit string to multiple state variables that are affected by the common prefix of the related queries. The remaining out-states are sampled uniformly at random. It follows that there are exactly $\eta(\tau) = \sum_{i=1}^q \ell - \mathsf{llcp}_b\left(M^i; M^1, \ldots, M^{i-1}\right)$ b-bit values in any transcript τ, that are sampled independently and uniformly. We thus have $\Pr\left[D_Y^{\mathscr{A}} = \tau\right] = 2^{-\eta(\tau)b}$ for any τ.

Let Ω_X be the set of all possible coins of the real world (i.e. the game $\diamond\mathbf{prfx}\text{-}\mathbf{R}_{\text{FKS}}^f$). We have that $|\Omega_X| = 2^{b2^b}$, because the only source of randomness in the ideal world is the sampling of the function f. Let $\mathsf{comp}_X(\tau) \subseteq \Omega_X$ be the set of all coins compatible with the transcript τ, i.e. the set of the functions $f \in \mathrm{Func}(2^b)$ that are capable of producing τ in the experiment with $\mathscr{A}$. We will compute the probability of seeing τ in the real world as $\Pr\left[D_X^{\mathscr{A}} = \tau\right] = |\mathsf{comp}_X(\tau)|/|\Omega_X|$. Note that a real-world oracle is completely determined by the underlying function f.

If $\tau \in \mathcal{T}_{\mathrm{good}}$, then every in-state $s_j^i = t_{j-1}^i \oplus \bar{M}_j^i$ that does not trivially collide with some other in-state $s_{j'}^{i'}$ due to common prefix of $\bar{M}_j^i$ and $\bar{M}_{j'}^{i'}$ must be distinct. The number of domain points of f that have an image determined by τ is easily seen to be $\eta(\tau) = \sum_{i=1}^q \ell - \mathsf{llcp}_b\left(M^i; M^1, \ldots, M^{i-1}\right)$. A compatible function f can therefore have arbitrary image values on the remaining $2^b - \eta(\tau)$ domain points. Thus we compute $|\mathsf{comp}_X(\tau)| = 2^{b\left(2^b - \eta(\tau)\right)}$ and

$$\Pr\left[D_X^{\mathscr{A}} = \tau\right] = \frac{|\mathsf{comp}_X(\tau)|}{|\Omega_X|} = \frac{2^{b\left(2^b-\eta(\tau)\right)}}{2^{b2^b}} = 2^{-\eta(\tau)b} = \Pr\left[D_Y^{\mathscr{A}} = \tau\right].$$

It follows that $\Pr\left[D_X^{\mathscr{A}} = \tau\right] / \Pr\left[D_Y^{\mathscr{A}} = \tau\right] = 1$ for every $\tau \in \mathcal{T}_{\mathrm{good}}$.

Bounding the Probability of a Bad Transcript in the Ideal World We can bound the probability of τ being bad (cf. (6.1)) by first bounding the probability of a non-trivial collision of an arbitrary but fixed pair of in-states $s_j^i, s_{j'}^{i'}$ (i.e. the event $s_j^i = s_{j'}^{i'}$ occurs) and then summing this probability for all possible values of $(i,j), (i',j')$ with $(i',j') \neq (i,j)$. Because this probability varies significantly, we will split all in-states into three classes, and bound probabilities of individual collisions between these classes.

We will associate to each in-state s_j^i a label stamp_j^i. We set $\mathsf{stamp}_j^i = \texttt{free}$ if $1 < j = \mathsf{llcp}_b\left(\bar{M}^i; \bar{M}^1, \ldots, \bar{M}^{i-1}\right) + 1 \leq m_i$ such that $m_{i^*} < j$ for some $i^* < i$. We will set $\mathsf{stamp}_i^1 = \texttt{initial}$ for $1 \leq i \leq q$ and $\mathsf{stamp}_j^i = \texttt{fixed}$ in the remaining cases.

Informally, we have $\mathsf{stamp}_j^i = \texttt{free}$ whenever the adversary forces $\mathsf{outer}\left(t_{j-1}^i\right) = Z_{j-m_{i^*}-1}^{i^*}$ by reusing exactly first $j-1$ blocks of a previous query $\bar{M}^{i^*}$ in the query $\bar{M}^i$ and sets $\bar{M}_j^i \neq \bar{M}_j^{i^*} = 0^b$. By doing this, $\mathscr{A}$ freely but non-trivially chooses $\mathsf{outer}\left(s_j^i\right) = \mathsf{outer}\left(s_j^{i*} \oplus \bar{M}_j^{i*} \oplus \bar{M}_j^i\right)$. Note that if the adversary puts $\bar{M}_j^i = \bar{M}_j^{i^*}$, this is not counted as a free state (the states will in fact be the same). We have $\mathsf{stamp}_j^i = \texttt{initial}$ for the initial in-state of every query.

As the condition (6.1) is symmetrical with respect to (i,j) and (i',j'), and as it cannot be satisfied if $(i,j) = (i',j')$, it can be rephrased as

$$\begin{gathered}\exists (1,1) \leq (i',j') < (i,j) \leq (q,\ell) \text{ such that:} \\ \mathsf{llcp}_b\left(\bar{M}^i; \bar{M}^1, \ldots, \bar{M}^{i-1}\right) < j \leq \ell,\ s_j^i = s_{j'}^{i'}.\end{gathered} \tag{6.2}$$

Doing so is without loss of generality, as each s_j^i with $j \leq \mathsf{llcp}_b\left(\bar{M}^i; \bar{M}^1, \ldots, \bar{M}^{i-1}\right)$ is identical with some previous state that has already been checked for collisions with $s_{j'}^{i'}$ for every possible (i',j'). In the further analysis, we will be working with (6.2) rather than with (6.1).

We now bound the probability of collision of an arbitrary pair of in-states $(s^i_j, s^{i'}_{j'}) = (t^i_{j-1} \oplus \bar{M}^i_j, t^{i'}_{j'-1} \oplus \bar{M}^{i'}_{j'})$ with $\mathsf{stamp}^i_j = \mathtt{fixed}$. We fix i to an arbitrary value and investigate the following three cases for j. In each case, we treat every $(i', j') < (i, j)$.

Case 1: $\mathsf{llcp}_b\left(\bar{M}^i; \bar{M}^1, \ldots, \bar{M}^{i-1}\right) + 1 < j \leq m_i$. Here, t^i_{j-1} is undetermined when the adversary issues the query $\bar{M}^i$. This implies that it will be independent from all $t^{i'}_{j'-1}$ for any $(i', j') < (i, j)$. The probability of the collision $t^i_{j-1} \oplus \bar{M}^i_j = t^{i'}_{j'-1} \oplus \bar{M}^{i'}_{j'}$ is easily seen to be 2^{-b}.

Case 2: $\max\left\{\mathsf{llcp}_b\left(\bar{M}^i; \bar{M}^1, \ldots, \bar{M}^{i-1}\right) + 1, m_i\right\} < j \leq \ell$. We have $\bar{M}^i_j = 0^b$ and $t^i_{j-1} = Z^i_{j-M^i} \| \mathsf{inner}\left(t^i_{j-1}\right)$. Although the adversary learns the value of $Z^i_{j-M^i}$ during the experiment, it is generated independently of all $s^{i'}_{j'}$ with $(i', j') < (i, j)$ (because $j + 1 > \mathsf{llcp}_b\left(\bar{M}^i; \bar{M}^1, \ldots, \bar{M}^{i-1}\right)$). Even if $\mathsf{stamp}^{j'}_{i'} \in \{\mathtt{free}, \mathtt{initial}\}$ and $\mathsf{outer}\left(s^{i'}_{j'}\right) = \alpha$ for some value α chosen by the adversary, the collision between $s^i_j = Z^i_{j-M^i} \| \mathsf{inner}\left(t^i_{j-1}\right)$ and $s^{i'}_{j'} = \alpha \| \mathsf{inner}\left(s^{i'}_{j'}\right)$ happens with probability 2^{-b}.

Case 3: $j = \mathsf{llcp}_b\left(\bar{M}^i; \bar{M}^1, \ldots, \bar{M}^{i-1}\right) + 1$. If $j = \mathsf{llcp}_b\left(\bar{M}^i, \bar{M}^{i'}\right) + 1$, the in-state $s^{i'}_{j'=j}$, call it a twin-state of s^i_j, cannot collide with s^i_j, as $j - 1 = \mathsf{llcp}_b\left(\bar{M}^i, \bar{M}^{i'}\right)$ implies both that $t^i_{j-1} = t^{i'}_{j-1}$ and that $\bar{M}^i_j \neq \bar{M}^{i'}_j$.

We further claim that $\mathsf{outer}\left(t^i_{j-1}\right)$ has not been set and revealed to the adversary by any previous output value. If there was an $i^* < i$ with $m_{i^*} \leq \mathsf{llcp}_b\left(\bar{M}^i, \bar{M}^{i^*}\right) = j - 1$ and $j \leq m_i$ then we would have $\mathsf{stamp}^i_j = \mathtt{free}$. If we had the same situation but with $j > m_i$ then $\bar{M}^i$ and $\bar{M}^{i^*}$ would be identical. So $\mathsf{outer}\left(t^i_{j-1}\right)$ has indeed not been revealed to $\mathscr{A}$, and for any non-twin in-state $s^{i'}_{j'}$, the probability of collision is at most 2^{-b} by a similar argument as in **Case 1**.

There are no more than $q\ell$ choices for (i, j) and no more than $q\ell$ possible (i', j') for every (i, j) so the overall probability that the condition (6.2) will be evaluated due to a pair of in-states with $\mathsf{stamp}^i_j = \mathtt{fixed}$ is at most $(q\ell)^2/2^b$.

If $\mathsf{stamp}^i_j = \mathtt{free}$, then $\mathsf{outer}\left(s^i_j\right)$ is under adversary's control. However the value of $\mathsf{inner}\left(t^i_{j-1}\right)$ is always generated at the end of the experiment. By a case analysis similar to the previous one we can verify that the probability of a collision due to a pair of in-states with $\mathsf{stamp}^i_j = \mathtt{free}$ is not bigger than 2^{-c}. It is apparent from the definition of a $\mathtt{free}$ in-state that there is at most one such in-state for each query. Having $q\ell$ in-states in total, there are at most $q(q\ell)$ pairs with $\mathsf{stamp}^i_j = \mathtt{free}$ and the probability of $\tau \in \mathcal{T}_{\text{bad}}$ due to such a pair is at most $q^2\ell/2^c$.

If $\mathsf{stamp}^i_j = \mathtt{initial}$ then a non-triviall collision between s^i_j and any other $\mathtt{initial}$ in-state is impossible. A collision with a non-$\mathtt{initial}$ state $s^{i'}_{j'}$ implies that $t^{i'}_{j'-1} = \bar{M}^{i'}_{j'} \oplus \bar{M}^i_1$. If $j' > m_{i'}$ or if there is some M^{i^*} with $m_{i^*} < j' <= \mathsf{llcp}_b\left(M^{i'}, \bar{M}^{i^*}\right) + 1$, then outer $\left(t^{i'}_{j'-1}\right)$ is known to the adversary. However inner $\left(t^{i'}_{j'-1}\right)$ is always generated at the end of the experiment. By a case analysis similar to the one we carried out earlier, it can be verified that the collision $s^i_1 = s^{i'}_{j'}$ occurs with probability no bigger than 2^{-c}. There is exactly one `initial` in-state in each query, so similarly as with the `free` in-states, the overall probability of a transcript being bad due to a pair with an `initial` in-state is at most $q^2\ell/2^c$. By summing all the partial collision probabilities we obtain that $\Pr\left[D^{\mathscr{A}}_Y \in \mathcal{T}_{bad}\right] \leq (q\ell)^2/2^b + 2q^2\ell/2^c$. □

6.7 Security of Full-State Keyed Duplex

For FKD, we prove the following result:

Theorem 6.5. *Let $b, r, c, k > 0$ be such that $b = r + c$ and $k \leq c$. Let $\mathscr{A}$ be an information-theoretic adversary that makes no more than N queries to p and p^{-1} in total, that makes no more than q* Initialize *queries, such that each such query is followed by no more than ℓ* Duplexing *queries with the total maximal multiplicity limited by μ. Let* FKD *be the scheme defined in Figure 6.5. Then,*

$$\mathbf{Adv}^{\mathbf{prfd}}_{\mathrm{FKD}}(\mathscr{A}) \leq \frac{(q\ell)^2}{2^b} + \frac{(q\ell)^2}{2^c} + \frac{\mu N}{2^k}.$$

The proof of Theorem 6.5 uses Lemma 6.6 to transform an FKD adversary into an FKS adversary, similarly as Bertoni et al. did in their analysis of the Duplex construction [BDPA11a, BDPA11b]. While this would be sufficient to prove the security of the Duplex construction, the bound induced solely by Lemma 6.6 suffers from a quantitative degradation: we would have that $\mathbf{Adv}^{\mathbf{prfd}}_{\mathrm{FKD}}(\mathscr{A}) \leq \mathbf{Adv}^{\mathbf{prfx}}_{\mathrm{FKS}}(\mathscr{B})$ for a $\mathscr{B}$ that makes $q\ell$ queries, resulting in a bound $\frac{2q^2\ell^4}{2^b} + \frac{2q^2\ell^3}{2^c} + \frac{\mu N}{2^k}$ according to Theorem 6.3.

In reality, there *will be* a quantitative gap between the security of FKD construction and that of FKS present, but it will be smaller. This is because an FKS adversary constructed from an FKD adversary issues queries of a specific structure which is far from general. In the following proof for FKD, we use this property; we define a class of "constrained adversaries" and adjust the proof of Lemma 6.4 to these adversaries.

Proof of Theorem 6.5. We define the game $\diamond$**prfd-R** by including the lines marked by $\diamond$ in the game **prfd-R** in Figure 6.6. By doing so, we replace the public permutation p by an independent secret permutation π in all the primitive calls made by the FKD. We use this game as an intermediate step in the analysis, and define

$$\mathbf{Adv}^{\diamond\mathbf{prfd}}_{\mathrm{FKD}}(\mathscr{A}) = \Pr\left[\mathscr{A}^{\diamond\mathbf{prfd\text{-}R}_{\mathrm{FKD}}} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathbf{prfx\text{-}I}_{\mathrm{FKD}}} \Rightarrow 1\right]$$

for the sake of tidiness of the proof.

We have that $\mathrm{FKS}^p_K = \mathrm{FKS}_0^{E^p_K}$ (see Figure 6.7). Abusing the Δ-notation for the PRP security games, it follows that

$$\begin{aligned}\mathbf{Adv}^{\mathbf{prfd}}_{\mathrm{FKD}}(\mathscr{A}) &= \Delta_{\mathscr{A}}\left(\mathbf{prfd\text{-}R}_{\mathrm{FKD}};\mathbf{prfd\text{-}I}_{\mathrm{FKD}}\right)\\ &= \Delta_{\mathscr{A}}\left(\mathbf{prfd\text{-}R}_{\mathrm{FKD}};\diamond\mathbf{prfd\text{-}R}_{\mathrm{FKD}}\right) + \Delta_{\mathscr{A}}\left(\diamond\mathbf{prfd\text{-}R}_{\mathrm{FKD}};\mathbf{prfd\text{-}I}_{\mathrm{FKD}}\right)\\ &\le \Delta_{\mathscr{B}}\left(p, E^p_K; p, \pi\right) + \Delta_{\mathscr{C}}\left(\diamond\mathbf{prfd\text{-}R}_{\mathrm{FKD}};\mathbf{prfd\text{-}I}_{\mathrm{FKD}}\right)\\ &\le \mathbf{Adv}^{\mathbf{prp}}_{E^p}(\mathscr{B}) + \mathbf{Adv}^{\diamond\mathbf{prfd}}_{\mathrm{FKS}}(\mathscr{C})\end{aligned}$$

for some adversary $\mathscr{B}$ that makes $q \cdot \ell$ construction queries that have total maximal multiplicity μ, and makes N queries to p and p^{-1}, and for some adversary $\mathscr{C}$ that makes q Initialize queries, such that each such query is followed by no more than ℓ Duplexing queries. This claim is proved by the same argument as in the proof of Theorem 6.3.

Andreeva et al. proved that $\mathbf{Adv}^{\mathbf{prp}}_{E^p}(\mathscr{B}) \le \frac{\mu N}{2^k}$ for any $\mathscr{B}$ [ADMA15]. In Corollary 6.7 we show that any FKD adversary $\mathscr{C}$ can be turned into a special "constrained" adversary $\mathscr{B}'$ against FKS that makes $q\ell$ queries such that each query induces no more than ℓ calls to the permutation, and such that we have

$$\mathbf{Adv}^{\mathbf{prfd}}_{\mathrm{FKD}}(\mathscr{B}) \le \mathbf{Adv}^{\mathbf{prfx}}_{\mathrm{FKS}}(\mathscr{B}').$$

In Lemma 6.8, we prove that $\mathbf{Adv}^{\mathbf{prfx}}_{\mathrm{FKS}}(\mathscr{B}') \le (q\ell)^2/2^b + (q\ell)^2/2^c$ for any such $\mathscr{B}'$. $\square$

We define the mapping $Q^b_{\mathrm{FKS}} : (\{0,1\}^{<b})^+ \to \{0,1\}^*$ that will be used for the remainder of the proof. For any $b > 0$ and for all $X_1, \ldots, X_n \in \{0,1\}^{<b}$ we let

$$Q^b_{\mathrm{FKS}}(X_1, \ldots, X_n) = \mathrm{pad}_b(X_1) \| \ldots \| \mathrm{pad}_b(X_{n-1}) \| X_n.$$

Lemma 6.6 (Duplexing lemma [BDPA11a])**.** *Let $b, r, c, k > 0$ be such that $b = r + c$ and $k \le c$. Let $p \in \mathrm{Perm}(2^b)$. Let $D = \mathrm{FKD}^p$ as defined in Figure 6.5. Let $i \in \mathbb{N}^+$, $K \in \{0,1\}^k$, $M^1, \ldots, M^i \in \{0,1\}^{<b}$ and $z_1, \ldots, z_i \in \{1, \ldots, r\}$. Then after executing $D.\mathrm{initialize}(K)$ followed by $D.\mathrm{duplexing}\left(M^j, z_j\right)$ for $1 \le j < i$, for the i^{th} duplexing query (M^i, z_i) we always have*

$$Z_i = D.\mathrm{duplexing}\left(M^i, z_i\right) = \mathrm{FKS}^p(K, Q^b_{\mathrm{FKS}}(M^1, \ldots, M^i), z_i).$$

Moreover, the mapping $Q^b_{\mathrm{FKS}} : (\{0,1\}^{<b})^+ \to \{0,1\}^*$ is injective.

Proof. We will show the first claim by induction. For $i = 1$, the internal state of FKD is updated to $t_1 = p\left((0^{b-k}\|K) \oplus \mathrm{pad}_b(M^1)\right)$, which is exactly the same as the state of FKS evaluated on M^1 only. Then both FKD and FKS output the same value $Z_1 = \mathsf{left}_{z_1}(t_1)$. For every $i > 1$, FKD updates its state to $t_i = p\left(t_{i-1} \oplus \mathrm{pad}_b(M^i)\right)$. By the induction argument, t_{i-1} is also the state of FKS after processing the first $i - 1$ padded blocks. Then the final state of FKS is easily seen to be t_i as well. The equality of outputs follows trivially.

To verify the injectivity of Q^b_{FKS}, we will show how to invert it. For any image $X = Q^b_{\mathrm{FKS}}(X_1, \ldots, X_n)$, we can start recovering the input arguments from the left to right.

We have $n = \lceil |X|/b \rceil$. While $|X| > b$, we keep removing the leftmost b bits of X and applying the inverse of pad_b to them to recover the next component X_i. What remains is the unpadded block X_n. □

The result of Lemma 6.6 can be used to reduce any FKD adversary to a constrained FKS adversary. Given an adversary $\mathscr{A}$ against FKD, we define the reduction $\mathscr{A}' = R_{\mathrm{FKS}}(\mathscr{A})$ as follows. To answer the j^{th} duplexing query (M^i_j, z^i_j) made by $\mathscr{A}$ after the i^{th} initialize call, $\mathscr{A}'$ queries its own oracle with $(Q^b_{\mathrm{FKS}}(M^i_1, \dots, M^i_j), z^i_j)$. $\mathscr{A}'$ simply forwards the output of $\mathscr{A}$ at the end of the experiment.

Corollary 6.7. *Let $\mathscr{A}$ be an adversary against* FKD *that makes q initialize calls and duplexes ℓ blocks after each initialization and $R_{\mathrm{FKS}}(\mathscr{A})$ the constrained FKS adversary as defined above. It follows from Lemma 6.6, that $\mathbf{Adv}^{\diamond\mathbf{prfd}}_{\mathrm{FKD}}(\mathscr{A}) \leq \mathbf{Adv}^{\diamond\mathbf{prfx}}_{\mathrm{FKS}}(R_{\mathrm{FKS}}(\mathscr{A}))$.*

For the rest of the proof, we will consider FKD adversaries that make *exactly* q Initialize queries, such that each of them is followed by *exactly* ℓ Duplexing queries. This is without loss of generality, as any adversary $\mathscr{A}$ that makes less queries can be used to construct another adversary $\mathscr{B}$ that uses all the resources and has the same advantage as $\mathscr{A}$.[6] We denote by $\mathcal{A}'_{q,\ell}$ the set of constrained adversaries against FKS, that were each induced by some FKD adversary that makes q initialize calls and duplexes ℓ blocks after each initialization:

$$\mathcal{A}'_{q,\ell} = \{R_{\mathrm{FKS}}(\mathscr{A}) \ : \ \mathscr{A} \text{ an FKD adversary with resources exactly } (q,\ell)\}.$$

Lemma 6.8. *Let $b, r, c > 0$ be such that $b = r + c$. Let* FKS *be the scheme defined in Figure 6.5. Then,*

$$\mathbf{Adv}^{\diamond\mathbf{prfd}}_{\mathrm{FKS}}(\mathscr{A}') \leq \frac{(q\ell)^2}{2^b} + \frac{(q\ell)^2}{2^c},$$

for any constrained adversary $\mathscr{A}' \in \mathcal{A}'_{q,\ell}$

Proof. We will to a large extent follow the notation and conventions from the proof of Lemma 6.4. We assume that for $1 \leq i \leq q\ell$, the query M^i is already padded and ends with a non-zero final b-bit block with $m_i = |M^i|_b$ being the number of b-bit blocks in the query and $M^i_1, \dots, M^i_{m_i} \xleftarrow{b} M^i$ being the blocks. The structure of the queries and the number of squeezed bits will however differ.

Any adversary $\mathscr{A}' \in \mathcal{A}'_{q,\ell}$ makes exactly $q\ell$ FKS queries, but these queries comprise at most $q\ell$ unique b-bit blocks. Moreover, these queries follow a certain pattern. We have that for every $1 \leq i \leq q$:

$$M^{\ell(i-1)+1} = M^{\ell(i-1)+1}_1 \text{ and } M^{\ell(i-1)+j} = M^{\ell(i-1)+j-1} \| M^{\ell(i-1)+j}_j \text{ for } 2 \leq j \leq \ell,$$

[6] $\mathscr{B}$ simply runs $\mathscr{A}$, forwards all its queries and makes extra random Duplexing queries to waste resources. At the end of experiment, $\mathscr{B}$ outputs whatever $\mathscr{A}$ outputs.

where all $M_j^{\ell(i-1)+j} \in \{0,1\}^b$ are non-zero (due to padding). Note that we have $m_{\ell(i-1)+j} = j$ for $1 \leq i \leq q$ and $1 \leq j \leq \ell$. For every query, $\mathscr{A}'$ asks for no more than r output bits.

Because $\mathscr{A}'$ now only squeezes one block per query, the extended queries are now identical with the original queries. I.e. we have for $1 \leq i \leq q$ and $1 \leq j \leq \ell$ that $\bar{M}^{\ell(i-1)+j} = M^{\ell(i-1)+j}$. The internal in-states s_j^i and out-states t_j^i are defined the same way as in the proof of Lemma 6.4.

The RP-RF Switch Recall the game $\diamond$**prfd-R**^{f} defined in the proof of Lemma 6.4. We have

$$\begin{aligned}\mathbf{Adv}_{\text{FKS}}^{\diamond\mathbf{prfx\text{-}R}}(\mathscr{A}') &= \Delta_{\mathscr{A}'}\left(\diamond\mathbf{prfx\text{-}R}_{\text{FKS}}; \diamond\mathbf{prfx\text{-}R}_{\text{FKS}}^f\right) + \Delta_{\mathscr{A}'}\left(\diamond\mathbf{prfx\text{-}R}_{\text{FKS}}^f; \mathbf{prfx\text{-}I}_{\text{FKS}}\right)\\ &\leq (q\ell)^2/2^b + \Delta_{\mathscr{A}'}\left(\diamond\mathbf{prfx\text{-}R}_{\text{FKS}}^f; \mathbf{prfx\text{-}I}_{\text{FKS}}\right).\end{aligned}$$

Although there are $q\sum_{j=1}^{\ell} j = q\ell(\ell+1)/2$ calls to π during the $\diamond$**prfx-R** game when played by $\mathscr{A}'$, the structure of the queries implies, that there will be exactly $q\ell$ calls to π with unique input. We obtain the claimed inequality using Lemma 2.6 and a similar reduction as in the proof of Lemma 6.4, except that now the RP-RF distinguisher $\mathscr{B}$ records every $(x, \pi(x))$ that it learns from its oracle to avoid wasteful queries with repeated input.

Patarin's Coefficient-H Technique This part of the proof relies heavily on the corresponding part of the proof of Lemma 6.4. We will show that

$$\Delta_{\mathscr{A}'}\left(\diamond\mathbf{prfx\text{-}R}_{\text{FKS}}^f; \mathbf{prfx\text{-}I}_{\text{FKS}}\right) \leq (q\ell)^2/2^c.$$

The games a constrained adversary $\mathscr{A}' \in \mathcal{A}'_{q,\ell}$ is trying to distinguish are $\diamond$**prfx-R**$_{\text{FKS}}^f$ and **prfx-I**$_{\text{FKS}}$. A transcript $\tau = (\bar{M}^1, \ldots, \bar{M}^{q\ell}, T^1, \ldots, T^{q\ell})$ is defined as in the proof of Lemma 6.4, where $T^{\ell(i-1)+j}$ holds all the $j+1$ out-states appearing due to $\bar{M}^{\ell(i-1)+j}$ (including the dummy state $t_{\ell(i-1)+j}^0$). We will also use the same definition of a bad state (see the expression (6.1)). This immediately gives us $\Pr[D_X = \tau] / \Pr[D_Y = \tau] = 1$ for any $\tau \in \mathcal{T}_{\text{good}}$ by a similar argument as in the proof of Lemma 6.4. The probability $\Pr\left[D_Y \in \mathcal{T}^{bad}\right]$ needs new investigation.

Bounding the Probability of a Bad Transcript in the Ideal World We define the three possible labels of in-states, `free`, `initial` and `fixed` in the same way as before and we will work with the re-expressed definition of a bad state (6.2). Since the definitions of `free`, `initial` and `fixed` states are unchanged, the probabilities of collision due to a pair of in-states $s_j^i, s_{i'}^{j'}$ with $\mathsf{stamp}_j^i = \texttt{free}$, $\mathsf{stamp}_j^i = \texttt{initial}$ and $\mathsf{stamp}_j^i = \texttt{fixed}$ do not change. The only thing that really changes is the final counting.

For any $1 \leq i \leq q$, the query $\bar{M}^{\ell(i-1)+1} = M_1^i$ consists of a single block. Thus it only induces a single in-state with $\mathsf{stamp}_{\ell(i-1)+1}^1 = \texttt{initial}$. Then for any $2 \leq j \leq \ell$, we have

$\mathsf{llcp}_b\left(\bar{M}^{\ell(i-1)+j}, \bar{M}^{\ell(i-1)+j-1}\right) = j-1$, so there is at most one new in-state induced by $\bar{M}^{\ell(i-1)+j}$ and unaffected by the common prefix with the previous queries. It is $s^j_{\ell(i-1)+j}$, and we always have $\mathsf{stamp}^j_{\ell(i-1)+j} = \texttt{free}$.

We see that, with respect to (6.2), there is exactly one state $s^j_{\ell(i-1)+j}$ in the query $M^{\ell(i-1)+j}$ that can cause a non-trivial collision, giving us a total amount of $q\ell$ possible tuples (i,j).

For every such state, we need to count all other states (visited by (i',j') in (6.2)) with which it can collide. For any $i' < i$, it suffices to check equality of $s^j_{\ell(i-1)+j}$ with all ℓ in-states induced by $\bar{M}^{\ell(i'-1)+\ell}$, as every other query $\bar{M}^{\ell(i'-1)+j'}$ is its prefix. For $i' = i$, it suffices to look at in-states induced by $\bar{M}^{\ell(i-1)+j-1}$. Thus for any state $s^j_{\ell(i-1)+j}$, there are no more than $q\ell$ unique states, with which it can collide. Using the collision probabilities from the proof of Lemma 6.4, we conclude that $\Pr\left[D_Y \in \mathcal{T}^{bad}\right] \leq (q\ell)^2/2^c$. $\square$

6.8 Full-State SpongeWrap and its Security

Our result from Sect. 6.7 can be used to prove security of modified, more efficient versions of existing Sponge-based AE schemes. As an interesting instance, we introduce Full-state SpongeWrap, a variant of the authenticated encryption mode SpongeWrap [BDPA11a, BDPA11b], offering improved efficiency with respect to processing of associated data.

6.8.1 Authenticated Encryption for Sequences of Messages

In this Section, we focus on authenticated encryption schemes that act on sequences of AD-message pairs, following the approach of Bertoni et al.[7] [BDPA11a, BDPA11b].

A nonce-based scheme for authenticated encryption of AD-message sequences (NSAE) is a pair $\Pi = (\mathcal{K}, W)$ where the key space $\mathcal{K}$ is a finite set and W is a deterministic stateful algorithm surfacing three interfaces:

- W.initialize : $\mathcal{K} \times \mathcal{N} \to \emptyset$. Calling this interface will initialize W with a secret key $K \in \mathcal{K}$ and a nonce N from the nonce space $\mathcal{N} \subseteq \{0,1\}^*$.
- W.wrap : $\mathcal{A} \times \mathcal{M} \to \mathcal{C}$. This interface inputs an AD-message pair (A, M) from the AD space $\mathcal{A} \subset \{0,1\}^*$ and the message space $\mathcal{M} \subset \{0,1\}^*$, and outputs a ciphertext-tag pair $(C,T) \in \{0,1\}^* \times \{0,1\}^\tau$, where $|C| = |M|$ and T is a τ-bit tag authenticating (A, M) and all the queries processed by W so far (since the last initialization call). We call τ stretch, or ciphertext expansion of Π.
- W.unwrap : $\mathcal{A} \times \{0,1\}^* \times \{0,1\}^\tau \to \mathcal{M} \cup \{\bot\}$. This interface accepts a triple (A, C, T) of AD, ciphertext and tag from their respective domains, and outputs either a message $M \in \mathcal{M}$ or an error symbol $\bot$.

[7]Bertoni et al. do not consider an explicit nonce as we do; rather, they require the header of the first wrapping call to be unique.

We require that W is initialized before making the first wrapping or unwrapping call. For a given key K, we will use W_K to refer to the corresponding keyed instance, omitting K from the list of inputs; that is, $W.\text{initialize}(K, N) = W_K.\text{initialize}(N)$.

For the correctness of an NSAE $\Pi = (\mathcal{K}, W)$, we require that for every key $K \in \mathcal{K}$, $N \in \mathcal{N}$, number of wrapping queries $q \in \mathbb{N}^+$, vectors of bits $RL \in (\{0,1\})^q$, vectors of AD $\mathbf{A} \in \mathcal{A}^q$, and vectors of plaintexts $\mathbf{M} \in \mathcal{M}^q$, the variable OK remains set to true throughout the execution of the following code:

```
1: OK ← true
2: $W_0$.initialize($K, N$), $W_1$.initialize($K, N$)        ▷ Note there are two instances of $W$
3: for $i \leftarrow 1$ to $q$ do
4:     $C, T \leftarrow W_{RL[i]}$.wrap($A[i], M[i]$)        ▷ $RL[i]$ chooses the "direction" of $i^{\text{th}}$ query
5:     $M' \leftarrow W_{(1-RL[i])}$.unwrap($A[i], C, T$)
6:     if $M[i] \neq M'$ then OK ← false
7: end for
```

In other words, a correct NSAE scheme allows two parties to exchange encrypted AD-message pairs, such that each party needs only a single instance of W for both encryption and decryption, and the pattern of (directions of the) communication can be arbitrary.

Security of Authenticated Encryption We follow the approach of Bertoni et al. [BDPA11a, BDPA11b] for defining the security of AE. We split the twofold security goal of AE into two separate requirements, privacy and authenticity. The formal definition of security of NSAE schemes is given in Definition 6.9.

Definition 6.9 (SPRIV and SAUTH AE security)**.** *Given an NSAE scheme* $\Pi = (\mathcal{K}, W)$ *with a ciphertext expansion* τ *and an adversary* $\mathscr{A}$*, we define the advantage of* $\mathscr{A}$ *in breaking the confidentiality of* Π *in a chosen plaintext attack (with help of the games* **spriv-*R*** *and* **spriv-*I*** *in Figure 6.9) as*

$$\mathbf{Adv}_{\Pi}^{\mathbf{spriv}}(\mathscr{A}) = \Pr[\mathscr{A}^{\mathbf{spriv}\text{-}\boldsymbol{R}_\Pi} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{spriv}\text{-}\boldsymbol{I}_\Pi} \Rightarrow 1].$$

We define the advantage of an $\mathscr{A}$ *in breaking the authenticity of* Π *in a chosen ciphertext attack (wit help of the game* **sauth** *in Figure 6.9) as*

$$\mathbf{Adv}_{\Pi}^{\mathbf{sauth}}(\mathscr{A}) = \Pr[\mathscr{A}^{\mathbf{sauth}_\Pi} \text{ forges}]$$

where "$\mathscr{A}$ forges*" denotes the event that any query to the* Forge *oracle returns a value different from* $\perp$*.*

If $\mathbf{Adv}_{\Pi}^{\mathbf{spriv}}(\mathscr{A}) \leq \epsilon$ *for all adversaries* $\mathscr{A}$ *that make* q *Init queries, and after each Init query do wrapping queries that induce at most* ℓ *permutation calls (including the initialization) and with total maximal multiplicity* μ*, and that make* N *direct queries to the public permutation then we way that* Π *is a* $(\epsilon, q, \ell, \mu, N)$*-SPRIV-secure NSAE scheme.*

If $\mathbf{Adv}_{\Pi}^{\mathbf{sauth}}(\mathscr{A}) \leq \epsilon$ *for all adversaries* $\mathscr{A}$ *that make* q *Init queries, and after each*

```
proc initialize   $\boxed{\textbf{spriv-R}_\Pi}$
$p \leftarrow_\$ \mathrm{Perm}(2^b)$
$K \leftarrow_\$ \mathcal{K}$
$\mathcal{X} \leftarrow \varnothing$
init $\leftarrow$ false

proc $p(x), p^{-1}(y)$

proc Init($N$)
if $N \in \mathcal{X}$ then
    return false
init $\leftarrow$ true
$\mathcal{X} \leftarrow \mathcal{X} \cup \{N\}$
$W^p$.initialize($K, N$)
return true

proc Wrap($A, M$)
if init = false then
    return $\perp$
$(C, T) \leftarrow W^p$.wrap($A, M$)
return $(C, T)$
```

```
proc initialize   $\boxed{\textbf{spriv-I}_\Pi}$
$p \leftarrow_\$ \mathrm{Perm}(2^b)$
$\mathcal{X} \leftarrow \varnothing$
init $\leftarrow$ false

proc $p(x), p^{-1}(y)$

proc Init($N$)
if $N \in \mathcal{X}$ then
    return false
init $\leftarrow$ true
$\mathcal{X} \leftarrow \mathcal{X} \cup \{N\}$
return true

proc Wrap($A, M$)
if init = false then
    return $\perp$
$(C, T) \leftarrow_\$ \{0,1\}^{|M|} \times \{0,1\}^\tau$
return $(C, T)$
```

```
proc initialize   $\boxed{\textbf{sauth}_\Pi}$
$p \leftarrow_\$ \mathrm{Perm}(2^b)$
$K \leftarrow_\$ \mathcal{K}$
$\mathcal{X} \leftarrow \varnothing$, $\mathcal{Y} \leftarrow \varnothing$
$t \leftarrow \Lambda$

proc $p(x), p^{-1}(y)$

proc Init($N$)
if $N \in \mathcal{X}$ then
    return false
$\mathcal{X} \leftarrow \mathcal{X} \cup \{N\}$
$t \leftarrow \Lambda \| N$
$W^p$.initialize($K, N$)
return true

proc Wrap($A, M$)
if $t = \Lambda$ then
    return $\perp$
$(C, T) \leftarrow W^p$.wrap($A, M$)
$t \leftarrow t \| (A, C, T)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{t\}$
return $(C, T)$

proc Forge($N, (A_1, C_1, T_1), \ldots, (A_n, C_n, T_n)$)
if $(N, (A_1, C_1, T_1), \ldots, (A_n, C_n, T_n)) \in \mathcal{Y}$ then
    return $\perp$
$\bar{W}^p$.initialize($K, N$)
for $i \leftarrow 1$ to $n$ do
    $M \leftarrow \bar{W}^p$.unwrap($A_i, C_i, T_i$)
return $M$
```

Figure 6.9 – **Two-requirement definition of security for a nonce-based scheme for authenticated encryption of AD-message sequences** $\Pi = (\mathcal{K}, W)$ with ciphertext expansion τ. The variable t is a vector over the set $\{0,1\}^* \cup (\{0,1\}^*)^3$.

Init query do wrapping queries that induce at most ℓ permutation calls (including the initialization), that make N direct queries to the public permutation, that make at most q_v Forge *queries such that no* Forge *query induces more than ℓ permutation calls, and that have the total maximal multiplicity μ (for all three types of construction queries), then we say that Π is a $(\epsilon, q, \ell, \mu, N, q_v)$-SAUTH-secure nonce-based AE scheme.*

Remark 5. *We can assume w.l.o.g. that every query to the* Forge *oracle is either a fresh nonce followed by a single AD-ciphertext-tag triplet or a sequence of the form $(N, (A_1, C_1, T_1), \ldots, (A_n, C_n, T_n))$ with $(N, (A_1, C_1, T_1), \ldots, (A_{n-1}, C_{n-1}, T_{n-1}))$ having been learned by the adversary from a sequence of previous wrapping queries.*

This is because a single AD-ciphertext-tag triplet (that is not trivially known to be correct) at the end of the sequence is enough to make the forgery valid. At the same time, correct unwrapping of the first non-trivial AD-ciphertext-tag triplet is a necessary condition of the success of the whole query. An adversary $\mathscr{A}$ who issues Forge *queries with more than one non-trivial triplet can thus always be used to construct another adversary $\mathscr{B}$ that adheres to the structure of the queries we have just described such that $\mathbf{Adv}^{\mathrm{FSW}}_{\mathbf{sauth}}(\mathscr{B}) \geq \mathbf{Adv}^{\mathrm{FSW}}_{\mathbf{sauth}}(\mathscr{A})$.*

6.8.2 Full-State SpongeWrap

The Full-State SpongeWrap (FSW) is a permutation mode for authenticated encryption of AD-message sequences as described in Sect. 6.8.1. It is parametrized by a b-bit permutation p, the maximal message block size r, the key size k, the nonce size n, and the tag size $\tau > 0$. We require that $k \leq b - r =: c$ and $n < r$. We denote an instance with all parameters fixed by FSW$[p, r, k, n, \tau]$.

The key space of FSW is $\mathcal{K} = \{0,1\}^k$ and the nonce space is $\mathcal{N} = \{0,1\}^n$. The FSW construction internally uses an instance of FKD to process the inputs block by block. To ensure domain separation of different stages of processing a wrap-query, we use three *frame bits* placed at the same position in each duplexing call to FKD as explained in Table 6.1.

label	value	usage
F_{N}	000	process nonce, derive initial mask of a query
F_{AM}	001	block of A and M inside query
F_{M}	010	block of M inside query
F_{A}	011	block of A inside query
$F_{\mathrm{AM}\vert}$	100	last block of A and M inside query
$\bar{F}_{\mathrm{AM}}$	101	last block of A and M, query ends, produces tag
$\bar{F}_{\mathrm{M}}$	110	last block of M, query ends, produces tag
$\bar{F}_{\mathrm{A}}$	111	last block of A, query ends, produces tag

Table 6.1 – **Labelling and usage of the frame bits within FSW.**

```
1: algorithm wrap($A, M$) (outline)
2:     while there are both AD and message bits to process do
3:         take $\le r$ bit block of $M$ and $\le c - 5$ bit block of $A$
4:         wrap the message block
5:         if both $A$ and $M$ end then
6:             produce tag using frame bits $\bar{F}_{\mathrm{AM}}$
7:         else if only $A$ ends or only $M$ ends then
8:             process the blocks using frame bits $F_{\mathrm{AM|}}$
9:         else
10:            process the blocks using frame bits $F_{\mathrm{AM}}$
11:        end if
12:    end while
13:    while there are message bits to process do
14:        take $\le r$ bit block of $M$
15:        wrap the message block
16:        if $M$ ends then
17:            produce tag using frame bits $\bar{F}_{\mathrm{M}}$
18:        else
19:            process the blocks using frame bits $F_{\mathrm{M}}$
20:        end if
21:    end while
22:    while there are AD bits to process do
23:        take $\le r + c - 5$ bit block of $A$, split it into $r$ bit and $c - 5$ bit parts
24:        if $A$ ends then
25:            produce tag using frame bits $\bar{F}_{\mathrm{A}}$
26:        else
27:            process the parts using frame bits $F_{\mathrm{A}}$
28:        end if
29:    end while
30:    prepare $r$ random bits for next query using frame bits $F_{\mathrm{N}}$
31: end algorithm
```

Figure 6.10 – **Outline of an FSW$[p, r, k, n, \tau]$ wrap/unwrap(A, M) query**

The main motivation of the FSW is concurrent absorption of message and AD to achieve maximal efficiency through minimizing the number of permutation calls made to process each wrap-query.

Since we can only process r bits of a message input at a time,[8] we can use the remainder of the state for the frame bits and a block of AD. This implies the lengths of message and AD blocks processed with each permutation call; $r + 1$ bits for padded message block, 3 frame bits and (having in mind that the input to FKD is always padded) this leaves us at most $(b - 1) - (r + 1) - 3 = c - 5$ bits for a block of AD.

To minimize the number of permutation calls made in all possible situations, we further specify special treatment for the wrap/unwrap queries with more AD blocks

[8]This is because the number of message bits that we can encrypt at a time is limited by the output size of FKD.

than message blocks. An informal outline of a wrap query is given in Algorithm 6.10. This outline nicely illustrates how the frame bits are used for domain separation.

We next give a complete algorithmic description of the FSW. To keep it compact, we introduce the following notation. For any $L \in \{0,1\}^{\leq r}$, $R \in \{0,1\}^{\leq c-5}$ and $F \in \{0,1\}^3$, we let

$$Q(L,F,R) = \text{pad}_{r+1}(L) \| F \| R. \tag{6.3}$$

Note that $r+4 \leq |Q(L,F,R)| \leq b-1$ for any L,F,R. We let $(L,R) = \mathsf{lsplit}(X,n)$ denote splitting a string $X \in \{0,1\}^*$ into two parts such that $L = \mathsf{left}_{\min(|X|,n)}(X)$ and $R = \mathsf{right}_{|X|-|L|}(X)$. In particular, for $n \geq |X|$ we have $(X,\varepsilon) = \mathsf{lsplit}(X,n)$. We will use the abbreviation $D.\text{dpx}(M,z)$ for the interface $D.\text{duplexing}(M,z)$ of an FKD instance D. The interfaces of $\text{FSW}[p,r,k,n,\tau]$ are defined in Algorithm 6.11.

A schematic depiction of how the wrap interface processes various types of inputs is given in Figures 6.12 and 6.13.

6.8.3 Security of FSW

The security of FSW is relatively easy to analyse, thanks to the result from Section 6.7. The main steps of the analysis are to show that FSW injectively maps its wrapping queries to a sequence of FKD queries, and then to reduce the security of FSW to the security of FKD.

Theorem 6.10. *Let $b,r,c,k,n,\tau > 0$ be such that $b = r + c$, $k \leq c$ and $n < r$. Let $\mathscr{A}$ be an adversary that makes q Init queries, and after each Init query does wrapping queries that induce at most ℓ permutation calls (including the initialization) and with total maximal multiplicity μ, and that makes N direct queries to the public permutation. Let $\mathscr{A}'$ be an adversary that makes q Init queries, and after each Init query does wrapping queries that induce at most ℓ permutation calls (including the initialization), that makes at most q_v* Forge *queries such that no* Forge *query induces more than ℓ permutation calls, and that has the total maximal multiplicity μ (for all three types of construction queries). Let* FSW *be the scheme defined in Figure 6.11. Then,*

$$\begin{aligned}
\mathbf{Adv}^{\mathbf{spriv}}_{\text{FSW}}(\mathscr{A}) \leq & \frac{(q\ell)^2}{2^b} + \frac{(q\ell)^2}{2^c} + \frac{\mu N}{2^k}, \\
\mathbf{Adv}^{\mathbf{sauth}}_{\text{FSW}}(\mathscr{A}') \leq & \frac{((q+q_v)\ell)^2}{2^b} + \frac{((q+q_v)\ell)^2}{2^c} + \frac{\mu N}{2^k} + \frac{q_v}{2^\tau}.
\end{aligned}$$

Proof. We start by defining the games $\diamond$**spriv-R** and $\diamond$**sauth** which are respectively variants of the games **spriv-R** and **sauth** created by replacing the instance D of FKD (internally used by FSW) by a stateful algorithm RO^r_{FKD}, which does not use p at all.

The state of RO^r_{FKD} consists of two variables, a list $\mathbf{S}$ over $\{0,1\}^*$ and an array $\mathcal{F}$. RO^r_{FKD} exposes the same interfaces as FKD: (1) RO^r_{FKD}.initialize() that initializes the list $\mathbf{S}$, to the empty list Λ, and (2) RO^r_{FKD}.duplexing(X,z) that, on input $X \in \{0,1\}^{<b}$

```
interface $W$.initialize($K, N$)
  $D$.initialize($K$)
  $S \leftarrow \mathrm{pad}_r(N)\|0\|F_{\mathrm{N}}\|0^{c-5}$
  $Z \leftarrow D.\mathrm{dpx}(S, r)$
end interface
```

```
interface $W$.wrap($A, M$)
  $M_1\|\ldots\|M_m \xleftarrow{r} M$
  $(A', A^*) \leftarrow \mathsf{lsplit}(A, m(c-5))$
  $A'_1\|\ldots\|A'_{a'} \xleftarrow{c-5} A'$
  $A^*_1\|\ldots\|A^*_{a^*} \xleftarrow{b-5} A^*$
  if $m = a' = a^* = 0$ then
    $T \leftarrow \varepsilon$
    $F \leftarrow \bar{F}_{\mathrm{A}}$
  end if
  for $i \leftarrow 1$ to $a' - 1$ do
    $C_i \leftarrow M_i \oplus Z$
    $Z \leftarrow D.\mathrm{dpx}(Q(M_i, F_{\mathrm{AM}}, A'_i), r)$
  end for
  if $0 < a' < m$ or $0 < a', a^*$ then
    $C_{a'} \leftarrow M_{a'} \oplus \mathsf{left}_{|M_{a'}|}(Z)$
    $Z \leftarrow D.\mathrm{dpx}(Q(M_{a'}, F_{\mathrm{AM}|}, A'_{a'}), r)$
  else if $0 < m = a'$ and $a^* = 0$ then
    $C_{a'} \leftarrow M_{a'} \oplus \mathsf{left}_{|M_{a'}|}(Z)$
    $T \leftarrow D.\mathrm{dpx}(Q(M_{a'}, \bar{F}_{\mathrm{AM}}, A'_{a'}), r)$
    $F \leftarrow \bar{F}_{\mathrm{AM}}$
  end if
  for $i \leftarrow a' + 1$ to $m - 1$ do
    $C_i \leftarrow M_i \oplus Z$
    $Z \leftarrow D.\mathrm{dpx}(Q(M_i, F_{\mathrm{M}}, \varepsilon), r)$
  end for
  if $a' < m$ then
    $C_m \leftarrow M_m \oplus \mathsf{left}_{|M_m|}(Z)$
    $T \leftarrow D.\mathrm{dpx}(Q(M_m, \bar{F}_{\mathrm{M}}, \varepsilon), r)$
    $F \leftarrow \bar{F}_{\mathrm{M}}$
  end if
  for $i \leftarrow 1$ to $a^* - 1$ do
    $(L, R) \leftarrow \mathsf{lsplit}(A^*_i, r)$
    $D.\mathrm{dpx}(Q(L, F_{\mathrm{A}}, R), 0)$
  end for
  if $a^* > 0$ then
    $(L, R) \leftarrow \mathsf{lsplit}(A^*_{a^*}, r)$
    $T \leftarrow D.\mathrm{dpx}(Q(L, \bar{F}_{\mathrm{A}}, R), r)$
    $F \leftarrow \bar{F}_{\mathrm{A}}$
  end if
  while $|T| < \tau$ do
    $T \leftarrow T\|D.\mathrm{dpx}(Q(\varepsilon, F, \varepsilon), r)$
  end while
  $Z \leftarrow D.\mathrm{dpx}(Q(\varepsilon, F_{\mathrm{N}}, \varepsilon), r)$
  $C \leftarrow C_1\|\ldots\|C_m$
  return $C, \mathsf{left}_\tau(T)$
end interface
```

```
interface $W$.unwrap($A, C, T$)
  $C_1\|\ldots\|C_m \xleftarrow{r} C$
  $(A', A^*) \leftarrow \mathsf{lsplit}(A, m(c-5))$
  $A'_1\|\ldots\|A'_{a'} \xleftarrow{c-5} A'$
  $A^*_1\|\ldots\|A^*_{a^*} \xleftarrow{b-5} A^*$
  if $m = a' = a^* = 0$ then
    $T' \leftarrow \varepsilon$
    $F \leftarrow \bar{F}_{\mathrm{A}}$
  end if
  for $i \leftarrow 1$ to $a' - 1$ do
    $M_i \leftarrow C_i \oplus Z$
    $Z \leftarrow D.\mathrm{dpx}(Q(M_i, F_{\mathrm{AM}}, A'_i), r)$
  end for
  if $0 < a' < m$ or $0 < a', a^*$ then
    $M_{a'} \leftarrow C_{a'} \oplus \mathsf{left}_{|C_{a'}|}(Z)$
    $Z \leftarrow D.\mathrm{dpx}(Q(M_{a'}, F_{\mathrm{AM}|}, A'_{a'}), r)$
  else if $0 < m = a'$ and $a^* = 0$ then
    $M_{a'} \leftarrow C_{a'} \oplus \mathsf{left}_{|C_{a'}|}(Z)$
    $T' \leftarrow D.\mathrm{dpx}(Q(M_{a'}, \bar{F}_{\mathrm{AM}}, A'_{a'}), r)$
    $F \leftarrow \bar{F}_{\mathrm{AM}}$
  end if
  for $i \leftarrow a' + 1$ to $m - 1$ do
    $M_i \leftarrow C_i \oplus Z$
    $Z \leftarrow D.\mathrm{dpx}(Q(M_i, F_{\mathrm{M}}, \varepsilon), r)$
  end for
  if $a' < m$ then
    $M_m \leftarrow C_m \oplus \mathsf{left}_{|C_m|}(Z)$
    $T' \leftarrow D.\mathrm{dpx}(Q(M_m, \bar{F}_{\mathrm{M}}, \varepsilon), r)$
    $F \leftarrow \bar{F}_{\mathrm{M}}$
  end if
  for $i \leftarrow 1$ to $a^* - 1$ do
    $(L, R) \leftarrow \mathsf{lsplit}(A^*_i, r)$
    $D.\mathrm{dpx}(Q(L, F_{\mathrm{A}}, R), 0)$
  end for
  if $a^* > 0$ then
    $(L, R) \leftarrow \mathsf{lsplit}(A^*_{a^*}, r)$
    $T' \leftarrow D.\mathrm{dpx}(Q(L, \bar{F}_{\mathrm{A}}, R), r)$
    $F \leftarrow \bar{F}_{\mathrm{A}}$
  end if
  while $|T'| < \tau$ do
    $T' \leftarrow T'\|D.\mathrm{dpx}(Q(\varepsilon, F, \varepsilon), r)$
  end while
  $Z \leftarrow D.\mathrm{dpx}(Q(\varepsilon, F_{\mathrm{N}}, \varepsilon), r)$
  $M \leftarrow M_1\|\ldots\|M_m$
  if $T = \mathsf{left}_\tau(T')$ then
    return $M$
  else
    return $\perp$
  end if
end interface
```

Figure 6.11 – **The NSAE scheme** FSW$[p, r, k, n, \tau]$. The instance internally uses an instance D of FKD$[p, r, k]$

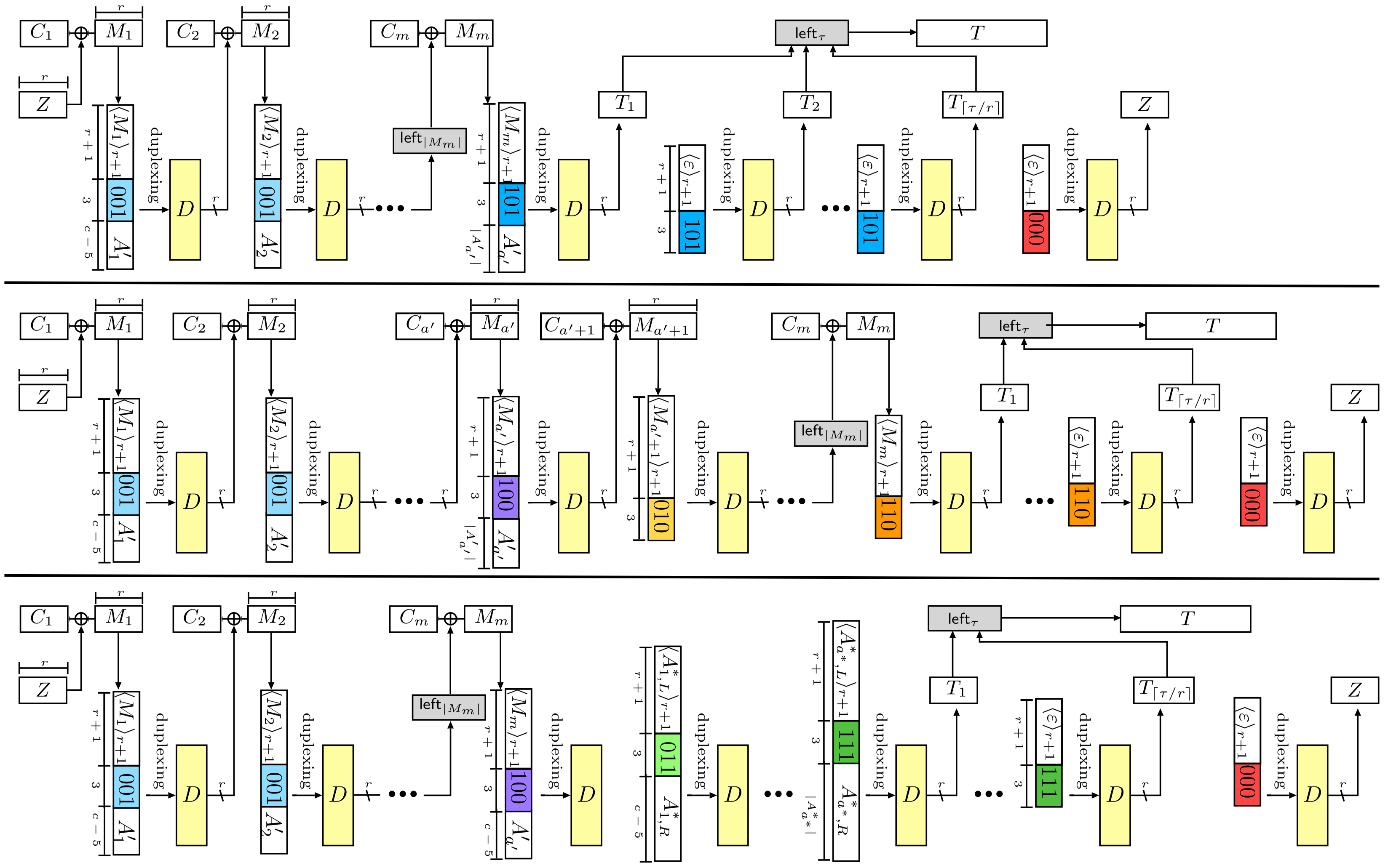

C_1
M_1
C_2
M_2
C_m
M_m
$C_{a'}$
$M_{a'}$
$C_{a'+1}$
$M_{a'+1}$
Z
r
$r+1$
3
$c-5$
$|A'_{a'}|$
$|A^*_{a^*}|$
$\langle M_1\rangle_{r+1}$ 001 A'_1
$\langle M_2\rangle_{r+1}$ 001 A'_2
$\langle M_m\rangle_{r+1}$ 101 $A'_{a'}$
$\langle M_{a'}\rangle_{r+1}$ 100 $A'_{a'}$
$\langle M_{a'+1}\rangle_{r+1}$ 010
$\langle M_m\rangle_{r+1}$ 110
$\langle M_m\rangle_{r+1}$ 100 $A'_{a'}$
$\langle A^*_{1,L}\rangle_{r+1}$ 011 $A^*_{1,R}$
$\langle A^*_{a^*,L}\rangle_{r+1}$ 111 $A^*_{a^*,R}$
$\langle\varepsilon\rangle_{r+1}$ 101
$\langle\varepsilon\rangle_{r+1}$ 110
$\langle\varepsilon\rangle_{r+1}$ 111
$\langle\varepsilon\rangle_{r+1}$ 000
duplexing
D
left$_{|M_m|}$
left$_\tau$
T_1
T_2
$T_{\lceil\tau/r\rceil}$
T

Figure 6.12 – **Initialization and wrapping processes of FSW$[\boldsymbol{p, r, k, n, \tau}]$.** FSW internally uses an instance of FKD$[p, r, k]$. The AD and message inputs A, M are partitioned as specified in Algorithm 6.11 (note that $A = A' \| A^*$). We let $\langle\cdot\rangle_n = \text{pad}_n(\cdot)$. The figure depicts from top to bottom: 1) The wrapping process when $\lceil |A|/(c-5)\rceil = \lceil |A|/r\rceil$; 2) The wrapping process when $\lceil |A|/(c-5)\rceil < \lceil |A|/r\rceil$;3) The wrapping process when $\lceil |A|/(c-5)\rceil > \lceil |A|/r\rceil$.

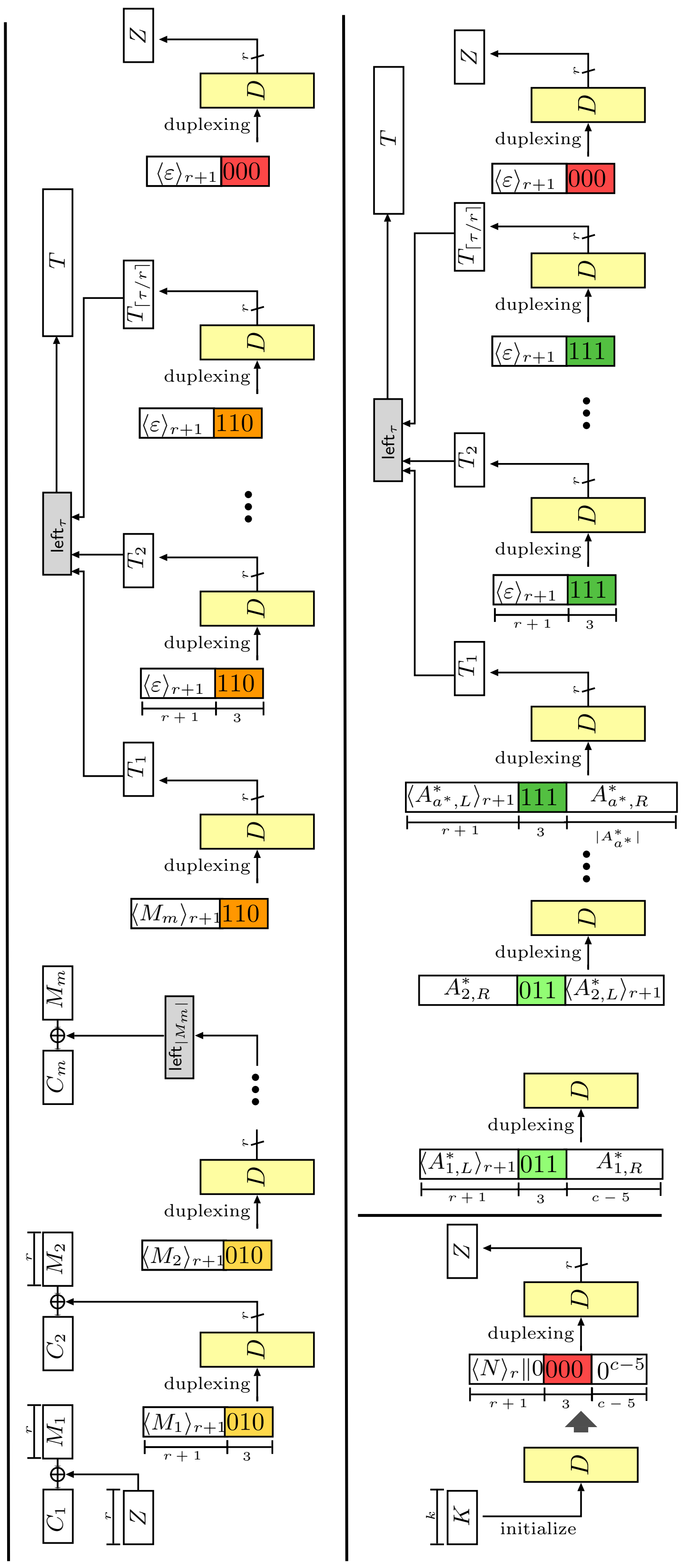


Figure 6.13 – **Initialization and wrapping processes of FSW$[p, r, k, n, \tau]$.** FSW internally uses an instance of FKD$[p, r, k]$. The AD and message inputs A, M are partitioned as specified in Algorithm 6.11 (note that $A = A'\|A^*$). We let $\langle\cdot\rangle_n = \mathrm{pad}_n(\cdot)$. The figure depicts from top to bottom: 1) The initialization process when $A = \varepsilon$; 2) The initialization process (left) and the wrapping process when $M = \varepsilon$ (right).

and a natural number z, first updates the list $\mathbf{S} \leftarrow \mathbf{S} \| X$, then if $\mathcal{F}[\mathbf{S}] = \bot$ sets $\mathcal{F}[\mathbf{S}] \leftarrow\$ \{0,1\}^r$, and outputs $\mathsf{left}_z\left(\mathcal{F}[\mathbf{S}]\right)$.

Note that the algorithm RO^r_{FKD} is completely equivalent with the pair of oracles Initialize and Duplexing in the game **prfd-I**. We use this to obtain the following inequalities:

$$\begin{aligned}\mathbf{Adv}^{\mathbf{spriv}}_{\mathrm{FSW}}(\mathscr{A}) &= \Delta_{\mathscr{A}}\left(\mathbf{spriv\text{-}R}_{\mathrm{FSW}}; \diamond\mathbf{spriv\text{-}R}_{\mathrm{FSW}}\right) + \Delta_{\mathscr{A}}\left(\diamond\mathbf{spriv\text{-}R}_{\mathrm{FSW}}; \mathbf{spriv\text{-}I}_{\mathrm{FSW}}\right) \\ &\leq \frac{(q\ell)^2}{2^b} + \frac{(q\ell)^2}{2^c} + \frac{\mu N}{2^k} + \Delta_{\mathscr{A}}\left(\diamond\mathbf{spriv\text{-}R}_{\mathrm{FSW}}; \mathbf{spriv\text{-}I}_{\mathrm{FSW}}\right),\end{aligned}$$

and

$$\begin{aligned}\mathbf{Adv}^{\mathbf{sauth}}_{\mathrm{FSW}}(\mathscr{A}') &= \Pr\left[\mathscr{A}'^{\mathbf{sauth}_{\mathrm{FSW}}} \text{ forges}\right] - \Pr\left[\mathscr{A}'^{\diamond\mathbf{sauth}_{\mathrm{FSW}}} \text{ forges}\right] + \Pr\left[\mathscr{A}'^{\diamond\mathbf{sauth}_{\mathrm{FSW}}} \text{ forges}\right] \\ &\leq \frac{((q+q_v)\ell)^2}{2^b} + \frac{((q+q_v)\ell)^2}{2^c} + \frac{\mu N}{2^k} + \frac{q_v}{2^\tau} + \Pr\left[\mathscr{A}'^{\diamond\mathbf{sauth}_{\mathrm{FSW}}} \text{ forges}\right].\end{aligned}$$

Both inequalities are proved by a simple reduction. An FKD adversary $\mathscr{B}$ can be constructed by using $\mathscr{A}$ (or $\mathscr{A}'$) as a subroutine. $\mathscr{B}$ simply simulates the **spriv-R** game (or the **sauth** game) using its own Initialize and Duplexing oracles in the place of D.initialize and D.duplexing calls. If $\mathscr{B}$ simulates the **spriv-R** game for $\mathscr{A}$, it outputs whatever $\mathscr{A}$ outputs. If $\mathscr{B}$ simulates the **sauth** game for $\mathscr{A}$, it outputs 1 if and only if $\mathscr{A}'$ forges.

If $\mathscr{B}$ is playing **prfd-R**$_{\mathrm{FKD}}$ game, then it perfectly simulates **spriv-R**$_{\mathrm{FSW}}$ for $\mathscr{A}$ and otherwise it perfectly simulates $\diamond$**spriv-R**$_{\mathrm{FSW}}$ (or it perfectly simulates **sauth**$_{\mathrm{FSW}}$ for $\mathscr{A}'$ and otherwise it perfectly simulates $\diamond$**sauth**$_{\mathrm{FSW}}$). If $\mathscr{B}$ uses $\mathscr{A}$as a subroutine, it will make q Initialize queries, while with $\mathscr{A}'$ it will make $(q + q_v)$ initialize queries. The bounds then follow from Lemma 6.5.

We now bound $\Delta_{\mathscr{A}}\left(\diamond\mathbf{spriv\text{-}R}_{\mathrm{FSW}}; \mathbf{spriv\text{-}I}_{\mathrm{FSW}}\right)$ and $\Pr\left[\mathscr{A}'^{\diamond\mathbf{sauth}_{\mathrm{FSW}}} \text{ forges}\right]$. We know that a unique sequence of a nonce and AD-message pairs yields unique sequence of RO^r_{FKD} queries thanks to Lemma 6.11.

We have that $\Delta_{\mathscr{A}}\left(\diamond\mathbf{spriv\text{-}R}_{\mathrm{FSW}}; \mathbf{spriv\text{-}I}_{\mathrm{FSW}}\right) = 0$. This is because the uniqueness of the nonces used in the Init queries implies that every FSW.wrap(A, M) query is processed using an RO^r_{FKD} with a unique internal variable $\mathbf{S}$. This implies that every ciphertext block and every tag will be produced using independent uniform bits.

It remains to show that $\Pr\left[\mathscr{A}'^{\diamond\mathbf{sauth}_{\mathrm{FSW}}} \text{ forges}\right] \leq q_v/2^\tau$. We first analyse the advantage of an adversary who only makes a single forgery attempt.

To forge, the adversary must produce a sequence $(N, (A_1, C_1, T_1), \ldots, (A_n, C_n, T_n))$ that passes the authentication check. Due to Remark 5, this can either be a fresh nonce followed by only (A_1, C_1, T_1) or N can be reused and $((C_1, T_1), \ldots, (C_{n-1}, T_{n-1}))$ were obtained from a sequence of wrapping queries $(N, (A_1, M_1), \ldots, (A_{n-1}, M_{n-1}))$ but (C_n, T_n) was not returned by any consequent wrapping query (A_n, M_n).

In the former case, with a fresh nonce, RO^r_{FKD} has a fresh list $\mathbf{S}$ when (A_1, C_1, T_1) is unwrapped and T_1 will be compared to τ random bits. The probability of a forgery is $2^{-\tau}$ in this case.

In the latter case, all the triplets $((A_1, C_1, T_1), \ldots, (A_{n-1}, C_{n-1}, T_{n-1}))$ are trivially successfully unwrapped. However freshness of (A_n, C_n, T_n) implies that either $C_n \neq C'_n$

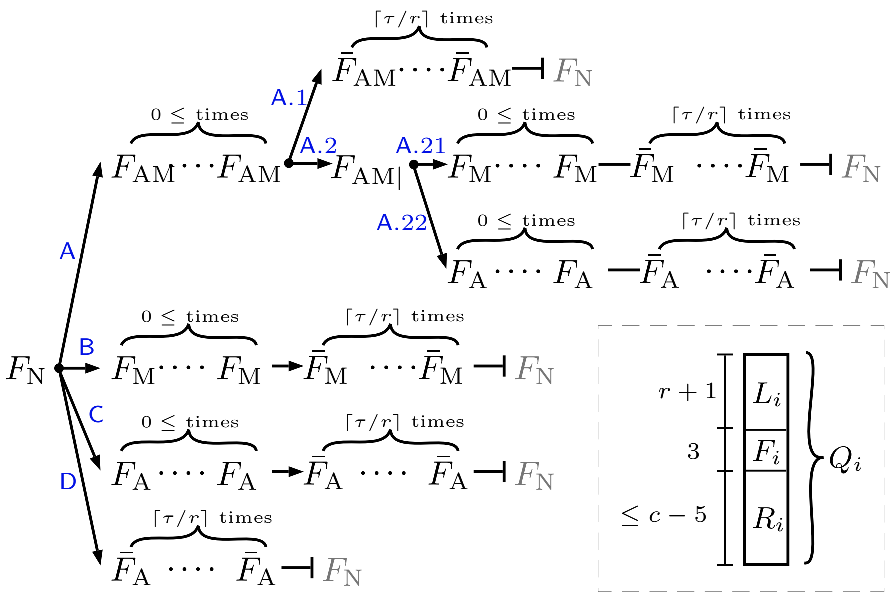

$\lceil\tau/r\rceil$ times
$\bar{F}_{\text{AM}}\cdots\bar{F}_{\text{AM}}\dashv F_{\text{N}}$
A.1
A.2
A.21
A.22
A
B
C
D
$0 \leq$ times
$F_{\text{AM}}\cdots F_{\text{AM}}$
$F_{\text{AM}|}$
$F_{\text{M}}\cdots F_{\text{M}} - \bar{F}_{\text{M}}\cdots\bar{F}_{\text{M}} \dashv F_{\text{N}}$
$F_{\text{A}}\cdots F_{\text{A}} - \bar{F}_{\text{A}}\cdots\bar{F}_{\text{A}} \dashv F_{\text{N}}$
F_{N}
$F_{\text{M}}\cdots F_{\text{M}} \to \bar{F}_{\text{M}}\cdots\bar{F}_{\text{M}} \dashv F_{\text{N}}$
$F_{\text{A}}\cdots F_{\text{A}} \to \bar{F}_{\text{A}}\cdots\bar{F}_{\text{A}} \dashv F_{\text{N}}$
$\bar{F}_{\text{A}}\cdots\bar{F}_{\text{A}} \dashv F_{\text{N}}$
$r+1$
3
$\leq c-5$
L_i
F_i
R_i
Q_i

Figure 6.14 – [**Framebit-sequences in a** FSW **query, and mapping to** FKD **queries.** The tree of all possible frame bits sequences that can occur when processing a single AD-message pair (top-left). The composition of an FKD query Q_i (bottom-right).

or $A_n \neq A\_n$ or $T_n \neq T'_n$ for the corresponding $(C'_n, T'_n) = \text{Wrap}(A'_n, M'_n)$ query, if such a wrapping query was made at all. If only $T_n \neq T'_n$, then T_n cannot be correct. In any other case, the tag T_n is compared to outputs of RO^r_{FKD} with either a fresh internal list **S** or with **S** that already occurred in the experiment but with fresh inputs. Thus T_n will be compared to τ uniform bits and the probability of a forgery is bounded by $2^{-\tau}$.

To obtain a more general result for an adversary that makes up to q_v verification queries, we use a similar reduction as one used by Bellare et al. [BGM04] and get $\Pr\left[\mathscr{A}'^{\diamond\mathbf{sauth}_{\text{FSW}}} \text{ forges}\right] \leq q_v/2^\tau$. □

Consider an instance $(\mathcal{K}, W) = \text{FSW}[p, r, k, n, \tau]$ of FSW. We let $\mathcal{Q}_W$ denote the function that maps a nonce and a sequence of AD-message pairs $(N, (A_1, M_1), \ldots, (A_n, M_n))$ to

$$\mathcal{Q}_W(N, (A_1, M_1), \ldots, (A_n, M_n)) \mapsto (Q_1, \ldots, Q_d)$$

such that $(Q_1, \ldots, Q_d) \in \{0,1\}^{<b}$ is the ordered sequence of all inputs to the D.duplexing calls made by the W.initialize and the subsequent queries to W.wrap during the processing of $(N, (A_1, M_1), \ldots, (A_n, M_n))$.

Lemma 6.11. *Let* $(\mathcal{K}, W) = \text{FSW}[p, r, k, n, \tau]$ *be an instance of* FSW *defined in Figure 6.11. Then the function* $\mathcal{Q}_W$ *is injective.*

Proof. We prove the injectivity of the mapping $\mathcal{Q}_W$ by showing how it can be inverted. We refer to the mapping Q defined in expression (6.3) to argue that every Q_i can be split into three strings L_i, F_i, R_i with $|L_i| = r + 1$, $|F_i| = 3$ and $|R_i| \leq c - 5$ just as depicted on the bottom-right of Figure 6.14. The main trick is to use the frame bits used in FSW to determine boundaries of wrapping queries and their logical parts. We will refer to the FKD queries $Q_1, \ldots, Q_d$ as "frames".

The nonce is contained in the very first frame with $F_1 = F_\mathrm{N}$ as $L_1 = \mathrm{pad}_r(N)\|0$. We can extract it, and discard the frame.

We can then recover the AD-message pairs (in the following just "pairs") from $\mathbf{Q} = (Q_1, \ldots, Q_d)$ in a left-to-right fashion. Any pair (A, M) is encoded in a subsequence of $\mathbf{Q}$ that starts by a frame with frame bits F_N and ends by a frame just before the next frame with frame bits F_N. Depending on the lengths of A and M, the pattern of frame bits between these boundary frames can differ as depicted in Fig. 6.14.

If both A and M are non-empty, we follow the edge marked as A. If there is the same number of r-bit blocks in M as there is of $c-5$ bit blocks in A, then we follow the path A.1. Otherwise we follow the path A.2 and then A.21 if there were fewer blocks in A than in M and the path A.22 if there were in turn more blocks in A than in M.

If $M \neq A = \varepsilon$, then we follow the path B; if $A \neq M = \varepsilon$ we follow the path C. In a special case, where both $A = M = \varepsilon$, we follow path D. We can see, that every possible case of relative = lengths of M and A in terms of blocks yields a distinct pattern of frame bit sequences.

Having identified which path in Fig. 6.14 we are following, we can recover A and M. Every frame Q_i with $F_i \in \{F_\mathrm{AM}, F_{\mathrm{AM}|}\}$ holds a padded block of M in L_i and an unpadded block of A in R_i. If $F_i = F_\mathrm{M}$, then there is a padded block of M in L_i and $R_i = \varepsilon$. If $F_i = F_\mathrm{A}$, then there is a padded block of A in L_i and another unpadded block of A in R_i. The frames with $F_i \in \{\bar{F}_\mathrm{AM}, \bar{F}_\mathrm{M}, \bar{F}_\mathrm{A}\}$ are used to produce the tag and are thus treated specially. The first frame with $\bar{F}_\chi$ holds data blocks and the following ones do not. If $\chi = \mathrm{AM}$, then there is a padded block of M in L_i and an unpadded block of A in R_i. If $\chi = \mathrm{M}$, then there is only a padded block of M in L_i. If $\chi = \mathrm{A}$ and we are not on path D then there is a padded block of A in L_i and a following unpadded block of A in R_i. If we are on path D then none of the frames holds any data, since both A and M are empty.

Once we extract all the blocks of A and M, we concatenate them all in the order in which they were extracted to obtain A and M. $\square$

Part II

Security Models

Chapter 7

Security of Online Authenticated Encryption

This chapter discusses the security of *online* authenticated encryption and its nonce-misuse resistance.

The work presented in this chapter is a result of joint work with Viet Tung Hoang, Reza Reyhanitabar and Phillip Rogaway that was published in CRYPTO 2015 [HRRV15].

Online encryption. This chapter is centred around *online* authenticated encryption. Onlineness is a functional property of an AE scheme, but we will see that it also impacts its security. When we speak of encryption being online we mean that it can be realized with constant memory while making a single left-to-right pass over the plaintext, writing out the ciphertext, also left-to-right, during that pass.

Investigating online encryption has a good reason: there are environments where it is needed. The designer of an FPGA or ASIC encryption/decryption engine might be unable to buffer more than a kilobyte of message. An application like SSH needs to instantaneously send a message every time a character is typed at the keyboard to emulate an interactive environment. Video-streaming services, such as Netflix need, to stream a video [Mia14] that is "played" as it is received, never buffering an excessive amount or incurring excessive delays. A software library might want to support an incremental encryption and decryption API.

Most of the AE schemes that target the simple nonce-based AE security notion (see Definition 2.7 or Definition 2.8) do have online encryption, e.g., 19 out of 20 the 2$^{\text{nd}}$ round CAESAR candidates that targeted nonce-based AE security had online encryption [Viz16].

Nonce Misuse Resistance and Onlineness. As indicated by Rogaway and Shrimpton [RS06b] and later reiterated by Fleischmann, Forler and Lucks, the security of nonce-based AE schemes is rather fragile in that it is possible—and routine—that all security will be forfeit once the nonces get repeated. The MRAE security notion (see Definition 2.10) was put forward by Rogaway and Shrimpton to limit the impact of nonce repetition to the bare minimum: having no adverse impact on authenticity, and the damage to privacy being limited to the detection of complete input-tuple repetitions.

While it is easy to construct MRAE schemes [RS06b] (see also Section 4.3), any such scheme must share a particular inefficiency: *its encryption can't be online.* The reason an MRAE scheme can't have online encryption is simple: the security definition demands that every bit of ciphertext depends on every bit of the plaintext, so one can't output the first bit of a ciphertext before reading the last bit of plaintext. Coupled with the constant-memory requirement, single-pass MRAE becomes impossible.

This limitation, the encryption of any MRAE-secure AE scheme being unavoidably offline,[1] led Fleischmann, Forler, and Lucks (FFL) to define a security notion [FFL12] that slots between NAE and MRAE. We call it OAE1. FFL claim that their notion captures the best-possible security for AE schemes with online encryption.

In this chapter, we investigate the security guarantees of OAE1. Based on our observations, we then formalize a security notion that we believe to come closer to the best possible security for online AE, and we show how to achieve it.

Organization of the Chapter. We give an overview of related work in Section 7.1 and list the contributions of this chapter in Section 7.2.

We recall the security notion OAE1 in Section 7.3 and in Section 7.4 we present definitional attacks and discuss shortcomings of the notion.

In Section 7.5, we then formalize our own take on the security of online AE, and in Section 7.6 we show how to achieve it using existing tools.

7.1 Related Work

The first security notion for nonce-misuse resistant security of online AE proposed by Fleischmann, Forler and Lucks (FFL) [FFL12] in 2012 induced most of this work. That result was in turn inspired by the notion of Online Ciphers by Bellare, Boldyreva, Knudsen and Namprempre (BBKN) [BBKN01] and by the work on nonce-misuse resistant AE by Rogaway and Shrimption [RS06b].

Fouque, Joux, Martinet, and Valette (FJMV) defined two notions for online AE, one for privacy and another for authenticity, as early as in 2003 [FJMV03]. However their aim was to formalize security against blockwise-adaptive attackers. In their setting, message space is the set of finite sequences of fixed-size blocks, encryption is online and probabilistic, but decryption is viewed as an atomic operation. In both notions, the adversary can mount blockwise-adaptive queries on the encryption oracle. OAE1 resembles a recasting of FJMV's notions with nonce-based AE syntax; this was later established formally by Endignoux and Vizár [EV16].

[1]For the sake of conciseness, we let "offline" mean "not online" in the rest of the chapter.

In 2004, Boldyreva and Taesombut (BT) [BT04] independently considered a similar setting to that of FJMV, in which the adversary can mount blockwise-adaptive attacks on schemes whose messages consist of fixed-size blocks. What differed from FJMV (and what is similar to OAE2) is that the adversary could query both encryption and decryption oracles in a blockwise manner. Their focus is on probabilistic encryption schemes under chosen-ciphertext-attack, but the full version of their paper [BT04, Section 6] also speaks of AE.

Tsang, Solomakhin, and Smith (TSS) [TSS09] were the first to sever the association of the blocksize of some underlying tool and the amount of data (i.e. bits) a user is ready to operate on, which is one of the key conceptual shifts needed to move beyond BBKN's and FFL's conceptions of online encryption. In their 2009 technical report TSS provided a definition based on this idea, and they give examples [TSS09, Section 8] of practical scenarios where the size of (segments of) plaintexts that need to be encrypted due to latency requirements are well below the block size of any secure primitive. Unlike OAE2, the security notion by TSS works with schemes in which there is ciphertext expansion only at the beginning and end. They do not authenticate the segmented plaintext but the string that is their concatenation. Our formalization of OAE2 also differs from the notion by TSS in that it lets the adversary run multiple, concurrent sessions of online encryption and decryption.

Bertoni, Daemen, Peeters, and Van Assche (BDPV) define an object very much like what we are calling an OAE2 scheme [BDPA11a] (their syntax of an AE scheme that acts on (ordered) sequences of AD-message pairs is the syntax of NSAE schemes defined in Section 6.8.1 of this thesis). The security notions for privacy and authenticity put forward by BDPV (similar to the SPRIV and SAUTH notions of Definition 6.9, except that a weak form of nonce-repetition is tolerated to the adversary by BDPV) resemble OAE2. In addition, the approach and correspondence with BDPV inspired our inclusion of vector-valued AD.

Andreeva, Bogdanov, Luykx, Mennink, Mouha, and Yasuda (ABLMMY) [ABL$^+$14a] study OAE definitions and schemes that are meant to withstand the release of unverified plaintext (RUP). Their motivations overlap with our own—a desire to support decrypting devices with insufficient memory to store the entire plaintext, or to allow prefixes of the decrypted plaintext to be revealed as soon as required, according to application's real-time needs. The authors define a variety of new security notions that capture uselessness of the prematurely released unverified plaintext for attacks against confidentiality, and those against authenticity of an AE scheme. Despite intersecting motivations, our definitional approach and theirs diverge. Unlike ABLMMY, our own notion works with user-segmented plaintext, avoids the use of knowledge extractors, and keeps the focus on encryption *and* decryption being online. When OAE2 is used in its intended manner, with reasonable segment-expansion τ, the notion is simultaneously stronger than RUP notions in the sense of ensuring that all decrypted segments *are* authentic; weaker than RUP notions in the sense that plaintext-extractors are in no way mandated; and incomparable in the sense that the extensive syntactic mismatch makes any meaningful implications or separations infeasible.

Around the time of the beginning of CAESAR competition, numerous AE designs appeared that targeted the OAE1-security, or weakened variants of it. In a relatively short amount of time, we witnessed a divergence between the rhetoric used to make security claims and the actual security guarantees provided by certain AE schemes, the actual guarantees themselves being spread over a wide spectrum. We discuss this further in Appendix B.1.

7.2 Contribution

We present two definitional attacks that help illustrate the actual (in)security captured by the OAE1 security notion of FFL. We also discuss broader shortcomings of the formalism used by FFL related to their choice of syntax.

We define segmented AE schemes, whose syntax improves over that of FFL in all points that we identified, and define the OAE2 security of such AE schemes. In our opinion, the OAE2 notion better approximates the phrase "best possible security under nonce-misuse" in the context of online schemes. At the same time, we are being clear about the inherent limitations of any online AE scheme.

We demonstrate the feasibility of OAE2-security by designing the CHAIN construction based on an existing tool, and proving that it is OAE2-secure. In addition, we map the deterioration of the OAE1-like guarantees offered by AE schemes that appeared around the start of the CAESAR competition.

7.3 Security Notion OAE1

While there are several variations of the online-AE security notion used throughout the AE literature (see Appendix B.1), they all spring from FFL [FFL12], who modelled the (supposedly) ideal online AE scheme by combining the definition of an online cipher from Bellare, Boldyreva, Knudsen, and Namprempre [BBKN01] with the definition of authenticity of ciphertexts (also called integrity of ciphertexts) [BR00, KY00, RBBK01]. In this section we recall the FFL definition, staying true to the original exposition as much possible, but necessarily deviating from it to correct an error. We call the (corrected) definition OAE1.

Syntax. For any positive integer n let $\mathsf{B}_n = \{0,1\}^n$ denote the set of n-bit *blocks*, let $\mathsf{B}_n^+ = \bigcup_{i=1}^{\infty}\{0,1\}^{i \cdot n}$ denote the set of all nonempty strings of n-bit blocks and let $\mathsf{B}_n^* = \mathsf{B}_n^+ \cup \{\varepsilon\}$. A *block-based AE scheme* is a triple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ where the key space $\mathcal{K}$ is a nonempty set with an associated distribution and where the encryption algorithm $\mathcal{E}$ and decryption algorithm $\mathcal{D}$ are deterministic algorithms with signatures $\mathcal{E}\colon \mathcal{K} \times \mathcal{H} \times \mathsf{B}_n^* \to \{0,1\}^*$ and $\mathcal{D}\colon \mathcal{K} \times \mathcal{H} \times \{0,1\}^* \to \mathsf{B}_n^* \cup \{\perp\}$. The set $\mathcal{H}$ associated to Π is the *header space*. FFL assume that it is $\mathcal{H} = \mathsf{B}_n^+ = \mathcal{N} \times \mathcal{A}$ with $\mathcal{N} = \mathsf{B}_n$ and $\mathcal{A} = \mathsf{B}_n^*$ the nonce space and AD space. The value n associated to Π is its *blocksize*.

Note that the message space $\mathcal{M}$ of Π must be $\mathcal{M} = \mathsf{B}_n^*$ and the blocksize n will play a central role in the security definition. We demand that $\mathcal{D}(K, N, A, \mathcal{E}(K, N, A, M)) = M$ for all $K \in \mathcal{K}$, $N \in \mathcal{N}$, $A \in \mathcal{A}$, and $M \in \mathsf{B}_n^*$.

To keep things simple, we further assume that the ciphertext expansion $|\mathcal{E}(K, H, M)| - |M|$ is a constant $\tau \geq 0$ rather than an arbitrary function of H and $|M|$. We let $\mathcal{E}_K^H = \mathcal{E}_K(H, M) = \mathcal{E}(K, H, M)$.

Security. Let $\mathrm{OPerm}[2^n]$ be the set of all length-preserving permutations π on B_n^* where i^{th} block of $\pi(M)$ depends only on the first i-blocks of M; more formally, a length-preserving permutation $\pi\colon \mathsf{B}_n^* \to \mathsf{B}_n^*$ is in $\mathrm{OPerm}[2^n]$ if for all $X, Y, Y' \in \mathsf{B}_n^*$ we have that

$$\mathsf{left}_{|X|}\left(\pi(X\|Y)\right) = \mathsf{left}_{|X|}\left(\pi(X\|Y')\right),$$

i.e. the first $|X|$ bits of $\pi(XY)$ and $\pi(XY')$ coincide. Despite its being infinite, one can endow $\mathrm{OPerm}[2^n]$ with a “uniform” distribution in a natural way: sampling a random $\pi \leftarrow\$ \mathrm{OPerm}[2^n]$ is equivalent to sampling $\pi_M \leftarrow\$ \mathrm{Perm}(2^n)$ for all $M \in \mathsf{B}_n^*$ and letting

$$\pi(M') = \pi_\varepsilon(M'_1)\|\pi_{M'_1}(M'_2)\|\ldots\|\pi_{M'_1\|\ldots\|M'_{|M'|_n-1}}(M'_{|M'|_n})$$

for every $M' \in \mathsf{B}_n^+$ where $M'_1\|\ldots\|M'_{|M'|_n} \overset{n}{\leftarrow} M'$, and letting $\pi(\varepsilon) = \varepsilon$. Note that for a small number of queries, a random $\pi \leftarrow\$ \mathrm{OPerm}[2^n]$ can be implemented efficiently with lazy sampling.[2]

Definition 7.1 (OAE1 security [FFL12]). *Given a block-based AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with blocksize n, header space $\mathcal{H}$ and a constant ciphertext expansion τ, and given an adversary $\mathscr{A}$, we define the advantage of $\mathscr{A}$ in breaking the OAE1 security of Π in a chosen ciphertext attack (with help of the games **oae1-*R*** and **oae1-*I*** in Figure 7.1) as*

$$\mathbf{Adv}_{\Pi}^{\mathbf{oae1}}(\mathscr{A}) = \Pr[\mathscr{A}^{\mathbf{oae1}\text{-}\boldsymbol{R}_\Pi} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{oae1}\text{-}\boldsymbol{I}_\Pi} \Rightarrow 1].$$

If $\mathbf{Adv}_{\Pi}^{\mathbf{oae1}}(\mathscr{A}) \leq \epsilon$ for all adversaries $\mathscr{A}$ whose running time is limited by t, and whose encryption and decryption query complexity is bounded by q_e and q_d respectively, and whose data complexity (in bits) in all queries is limited by σ then we say that Π is a $(\epsilon, t, q_e, q_d, \sigma)$-secure OAE1 scheme.

As usual, the exact resource parameters of the adversary may be adjusted to suit the analysis of a particular scheme. We can also speak of $\mathrm{OAE1}[n]$ security to emphasize the central role in defining security of the scheme's blocksize n.

[2]The implementation represents π as an initially empty 2^n-ary tree, whose vertices correspond to (partially defined) elements of $\mathrm{Perm}(2^n)$ and whose edges are labelled with elements of B_n. To evaluate a query $\pi(M_1\|\ldots\|M_m)$, one applies the permutation at the root of the tree to M_1 to get C_1, then follows the edge labelled with M_1 and applies the corresponding permutation to M_2 to get C_2, follows the edge labelled with M_2 and so on, and returns $C_1\|\ldots\|C_m$. Whenever an edge does not exist, it is created together with the corresponding vertex which is initialized to an undefined permutation, and then lazily sampled according to the queries.

```
proc initialize                      $\boxed{\textbf{oae1-R}_\Pi}$
$K \leftarrow\$ \mathcal{K}$
$\mathcal{Y} \leftarrow \emptyset$

proc Enc$(H, M)$
$C \leftarrow \mathcal{E}(K, H, M)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C)\}$
return $C$

proc Dec$(H, C)$
if $(H, C) \in \mathcal{Y}$ then
   return $\bot$
return $\mathcal{D}(K, H, C)$
```

```
proc initialize                      $\boxed{\textbf{oae1-I}_\Pi}$
for $H \in \mathcal{H}$ do $\pi_H \leftarrow\$ \mathrm{OPerm}[2^n]$
for $(H, M) \in \mathcal{H} \times \mathsf{B}_n^*$ do $R_{H,M} \leftarrow\$ \{0,1\}^\tau$

proc Enc$(H, M)$
return $\pi_H(M) \| R_{H,M}$

proc Dec$(H, C)$
return $\bot$
```

Figure 7.1 – **OAE1 security.** Defining security for a block-based AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with header space $\mathcal{H}$, blocksize n, and ciphertext expansion τ.

Deviation from the original definition. Definition 7.1 effectively says that, with respect to privacy, a ciphertext must resemble the image of a plaintext under a random online permutation (tweaked by the nonce and AD) followed by a τ-bit random string (the authentication tag). However, the original definition from FFL [FFL12, Definition 3] only modelled ciphertexts as images under a random online permutation, with no tag following. Such a definition does not make sense, as $\mathcal{E}$ must be length-increasing to provide authenticity. Necessarily, the scheme proposed by FFL (which did output a *tagged* ciphertext) could not conform with their definition. Definition 7.1 is a result of discussions between the author of this dissertation and the co-authors of the corresponding publication [HRRV15], and of checking with one of the FFL authors [Luc14].

LCP leakage. We say that a block-based AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with blocksize n is LCP[n] (for "longest common prefix") if for all $K \in \mathcal{K}$, $H \in \mathcal{H}$ and $M, M' \in \mathsf{B}_n^*$ we have

$$\mathsf{llcp}_n\left(M, M'\right) = \mathsf{llcp}_n\left(\mathcal{E}_K(H, M), \mathcal{E}_K(H, M')\right).$$

While all schemes we know claiming to be OAE1[n] are also LCP[n], an OAE1[n]-secure scheme isn't *necessarily* LCP[n]. This is because the requirement for OAE1[n] security is to be computationally close to an object that is LCP[n], and something being computationally close to an object with a property P doesn't mean it has property P.

Indeed it is easy to construct an artificial counterexample; for example, we can start with an OAE1[n]-secure scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ that is LCP[n], and define $\bar{\Pi} = (\mathcal{K} \times \{0,1\}^n, \bar{\mathcal{E}}, \bar{\mathcal{D}})$ with

$$\bar{\mathcal{E}}((K, K'), H, M) = \begin{cases} \mathcal{E}\left(K, H, \mathsf{left}_n(M) \,\|\mathsf{reverse}(\mathsf{right}_{|M|-n}(M))\right) & \text{if } \mathsf{left}_n(M) = K' \\ \mathcal{E}\left(K, H, M\right) & \text{otherwise} \end{cases}$$

where the function $\mathsf{reverse}(X)$ takes a string X and returns its bits in reversed order. OAE1 security of $\bar{\Pi}$ is only slightly degraded; if the adversary guesses K', $\bar{\mathcal{E}}$ will no longer behave as an online permutation, but guessing K' is unlikely. The scheme $\bar{\Pi}$ is no longer LCP$[n]$, because for each key, there exist messages on which the common prefix is not preserved.

Despite such counterexamples, any OAE1$[n]$-secure scheme must be *close* to being LCP$[n]$, in the sense that it will preserve the LCP$[n]$ property for an overwhelming majority of the inputs. Fix a block-base AE scheme Π as defined above and consider an adversary $\mathscr{A}$ that is given an oracle $\mathcal{E}_K(\cdot,\cdot)$ for $K \twoheadleftarrow\!\$\ \mathcal{K}$. Consider $\mathscr{A}$ to be *successful* if it outputs $H \in \mathcal{H}$ and $X, Y, Y' \in \mathsf{B}_n^*$ such that

$$\mathsf{llcp}_n\left(\mathcal{E}_K^H(XY), \mathcal{E}_K^H(XY')\right) < |X|_n$$

(i.e., the adversary found non-LCP behavior).

Let $\mathbf{Adv}_{\Pi}^{\mathbf{lcp}}(\mathscr{A})$ be the probability that $\mathscr{A}$ is successful. Then it's easy to transform $\mathscr{A}$ into an equally efficient adversary $\mathscr{B}$ (in the sense that $\mathscr{B}$ will have the same data and query complexity as $\mathscr{A}$ and will run in similar time) for which $\mathbf{Adv}_{\Pi}^{\mathbf{oae1}}(\mathscr{B}) = \mathbf{Adv}_{\Pi}^{\mathbf{lcp}}(\mathscr{A})$. Because of this, there is no real loss of generality, when discussing OAE1$[n]$ schemes, to assume them LCP$[n]$. In the next section we will do so.

7.4 Shortcomings of OAE1

In this section, we give our critique of the OAE1 notion. We first describe two definitional attacks on OAE1. We call them the trivial attack and the CPSS attack. These attacks are definitional in the sense that they cannot be used to invalidate OAE1-security of a block-based AE scheme, but instead apply to every OAE1-secure scheme, and even to the idealized reference object implemented by the game **oae1-I**.

We then further discuss certain characteristics of the OAE1 notion that we find ill-conceived. These observations are then applied when we define our own notion in Section 7.5.

7.4.1 Definitional Attacks

The trivial attack. We first observe that as the blocksize n decreases, OAE1 becomes weaker: an adversary that has the ability to ask chosen-plaintext queries can *decrypt* the ciphertext of an arbitrary m-block plaintext with $(2^n - 1)m$ encryption queries.

More precisely, we claim that an adversary that has access to a (properly keyed) encryption oracle Enc and is given a ciphertext $C = \mathcal{E}_K^H(M)$ with M sampled from B_n^m with an arbitrary distribution can always recover M with $(2^n - 1)m$ encryption queries.

We now describe what we call the *trivial* attack. Fix a ciphertext $C = C_1 \cdots C_m\, T = \mathcal{E}_K(H, M)$ with $C_i \in \mathsf{B}_n$. Using just the encryption oracle Enc, we want to recover C's plaintext $M = M_1 \cdots M_m$ with $M_i \in \mathsf{B}_n$. The attack proceeds as described in Figure 7.2. Informally, it recovers the i^{th} plaintext block, by iterating over the values $M_i \in \mathsf{B}_n$ and

```
algorithm TrivialAttack($H, C\|T$)
    $C_1\|\ldots\|C_m \xleftarrow{n} C$
    $M \leftarrow \varepsilon$
    for $i \leftarrow 1$ to $m$ do
        for $M_i \in \mathsf{B}_n \backslash \{1^n\}$ do
            query $C' \leftarrow \mathrm{Enc}(H, M)$
            if $C' = C_1\|\ldots\|C_i$ then
                $M \leftarrow M\|M_i$
                break
            end if
        end for
        if $|M|_n < i$ then
            $M \leftarrow M\|1^n$
        end if
    end for
    return $M$
end algorithm
```

Figure 7.2 – **The trivial attack** against a block-based AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with blocksize n.

encrypting $M_1 \cdots M_{i-1}\ M_i$ until one reply matches $C_1 \cdots C_i$ or there's only a single value M_i remaining. It is easy to verify that the trivial attack works with the worst-case complexity of $(2^n - 1)m$ encryption queries, even if the Enc oracle is implemented as in the game **oae1-I**. This attack is therefore unavoidable whenever n is small and the adversary can repeat the headers.

The CPSS attack. Even with the trivial attack taken into account, one might hope for OAE1 security as long as the blocksize is fairly large, like $n = 128$. We dash this hope by describing another header-repeating attack, one we call the *chosen-prefix / secret-suffix* (CPSS) attack, which works for any blocksize. The attack is simple, yet devastating. It is inspired by the well-known BEAST (Browser Exploit Against SSL/TLS) attack [DR11].

Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a block-based AE scheme with blocksize n satisfying LCP[n]. We consider a setting where messages M that get encrypted can be logically divided into a prefix P that is controlled by an adversary, then a suffix S that is secret, fixed, and not under the adversary's control, i.e. where $M = P\|S$. The goal of the adversary is to learn S.

More formally, the adversary gets access to a special oracle defined as $\mathrm{Enc}'(H, P) = \mathcal{E}_K^H(P\|S)$ for any P, for a properly sampled secret key K and for an S sampled from $\{0,1\}^{|S|}$ according to some distribution. The corresponding game is sketched in Figure 7.3.

To be realistic, we insist that the length of P be a multiple of b bits for some small

```
proc initialize
$K \leftarrow_{\$} \mathcal{K}$
$S \leftarrow_{\$} \mathcal{X}(\{0,1\}^{|S|})$

proc $\mathrm{Enc}'(H, P)$
return $\mathcal{E}(K, H, P\|S)$

proc finalize
return $S' = S$
```

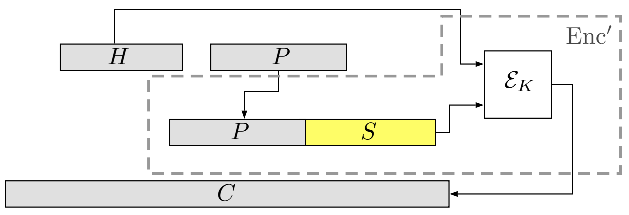


Figure 7.3 – **The setting of the CPSS attack.** The security game is parameterized by a block-based scheme Π with a block size n, a secret size $|S|$ and a distribution $\mathcal{X}(\{0,1\}^{|S|})$. The goal of the adversary is to output $S' = S$ at the end of the game. The game is outlined on the left, the special encryption oracle illustrated on the right.

positive integer b. This is assumed for S too. Typically P and S must be byte strings, whence $b = 8$; for concreteness, let us assume this is the case. Also for concreteness, assume a blocksize of $n = 128$ bits. Assume further that $\mathcal{E}$ can in fact operate on arbitrary byte-length strings, but suffers LCP leakage on block-aligned prefixes (this is what happens if one pads and then applies an OAE1-secure scheme). Finally, assume $|S|$ is a multiple of the blocksize. We claim that an adversary can recover the secret S with $|S|_b \cdot (2^b)$ encryption queries, irrespective of the blocksize n and the distribution with which S is sampled.

To recover S, the adversary proceeds as follows. First it selects an arbitrary string P^1 whose byte length is one byte shorter than p blocks, for an arbitrary $p \geq 1$. (For example, it would be fine to have $P^1 = 0^{120}$.) The adversary queries $C^1 = \mathrm{Enc}'(H, P^1) = \mathcal{E}_K^H(P^1\|S)$. This will be used to learn S_1, the first byte of S. To do so, the adversary queries $C^{1,B} = \mathrm{Enc}'(H, P^1\|B) = \mathcal{E}_K^H(P_1\|B\|S)$ for all-but-last one-byte values B (i.e. 255 queries). Due to LCP leakage, there will be at most one value of B for which we will have $\mathsf{llcp}_n\left(C^1, C^{1,B}\right) = p$: the one with $B = S_1$. If there is none, S_1 is equal to the only unqueried value of B. At this point the adversary knows the first byte of S, and has spent 256 queries to get it.

Now the adversary wants to learn S_2, the second byte of S. It selects an arbitrary string P^2 that is *two* bytes short of p blocks, for any $p \geq 1$. The adversary makes the query $C^2 = \mathrm{Enc}(H, P^2) = \mathcal{E}_K^H(P^2\|S)$; and it then queries $C^{2,B} = \mathrm{Enc}'(P^2\|S_1\|B) = \mathcal{E}_K^H(P^2\|S_1\|B\|S)$ for all-but-last one-byte values B. Due to LCP leakage and the fact that we have recovered the first byte S_1 of S already, at most one of these 255 values will give $\mathsf{llcp}_n\left(C^2, C^{2,B}\right) = p$, which will allow us to recover S_2, as before with S_1. At this point the adversary knows S_2, the second byte of S. It has used 256 more queries to get this.

Continuing in this way, the adversary recovers all of S in $256 \cdot |S|/8$ queries. In general, we need $2^b \cdot |S|_b$ queries to recover S, as we claimed. The general CPSS attack

```
1:  algorithm CPSS($|S|$)
2:      $p \leftarrow \lceil |S|/n \rceil$
3:      Pick an $H \in \mathcal{H}$
4:      $S' \leftarrow \varepsilon$
5:      for $i \leftarrow 1$ to $|S|/b$ do
6:          Pick a $P^i \in \{0,1\}^{p \cdot n - i \cdot b}$
7:          $C^i \leftarrow \mathrm{Enc}'(H, P^i)$
8:          $S_i \leftarrow \bot$
9:          for $B \in \{0,1\}^b \backslash \{1^b\}$ do
10:             $C^{i,B} \leftarrow \mathrm{Enc}'(H, P^i \| B)$
11:             if $\mathsf{llcp}_n\left(C^{i,B}, C^i\right) = p$ then $S_i \leftarrow B$
12:         end for
13:         if $S_i = \bot$ then $S_i \leftarrow 1^b$
14:         $S' \leftarrow S' \| S_i$
15:     end for
16:     return $S'$
17: end algorithm
```

Figure 7.4 – **The CPSS attack** against a block-based AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with blocksize n, and data granularity of b bits.

is described in pseudocode in Figure 7.4.

Note that the CPSS attack also works if the values that prefix S are not completely chosen by the adversary. It is enough that it be a known, fixed value, followed by the byte string that the adversary can fiddle with. That is, the attack applies when the adversary can manipulate a portion R of values $L\|R\|S$ that get encrypted, where L is known and S is not.[3]

How practical? It is not uncommon to have protocols where there is a predictable portion L of a message, followed by an adversarially mutable portion R specifying details, followed by further information S, some or all of which is sensitive. This happens in HTTP, for example, where the first portion of a request specifies a method, such as `GET`, the second specifies the requested resource, such as `/img/scheme.gif/`, and the final portion encodes information such as the HTTP version number, an end-of-line character, and a secret session cookie. If an LCP-leaking encryption scheme is used in such a setting, one is asking for trouble: the session cookie may be recovered by an adversary.

We do not suggest that LCP leakage will always lead to a real-world break. But if giving adversaries the ability to manipulate the middle portion R of plaintexts $L\|R\|S$ is sufficient for the nonce misuse to result in a practical attack, one has strayed very far

[3]A variant of this attack works even if the encrypted messages are of the form $L\|P\|R\|S$, where the adversary knows L, R and only controls P.

indeed from genuine misuse-resistance.

MRAE and CPSS. In Appendix B.2 we show that any MRAE-secure AE scheme resists the CPSS attack; establishing formally that there *is* a significant gap between the nonce misuse resistance guaranteed by OAE1 and that guaranteed by MRAE.

7.4.2 Broader OAE1 Critique

The CPSS attack suggests that it is inaccurate to label OAE1-security as nonce-misuse resistance, because OAE1-secure schemes deployed in common protocols are susceptible to realistic nonce-reusing attacks. In this section we further examine OAE1 and identify issues which we consider to be more fundamental. Because of these, the definition of OAE1 does, in our opinion, fail to capture the intuition about what something called "online-AE" ought to do.

The blocksize should not be a scheme-dependent constant. The motivation for making an AE scheme online is that in certain applications, the implementation of encryption can only buffer *some* limited number of bits of the plaintext at a time, be it due to the constraints of the platform or latency requirements. This in turn implies that the ciphertext must necessarily be composed of segments of *some* limited size, such that i^{th} ciphertext segment only depends on the first i corresponding plaintext segments.

OAE1[n] simply enforces "*some*" to be n, and demands that the i^{th} *block* depends only on the first i *blocks* of plaintext. Each of these blocks has a *fixed* blocksize, some number n associated to the scheme *and* its security definition. This implies that we always have to buffer *exactly* n bits to output the next segment of the ciphertext. It is not clear if this fixed amount of buffering is done as a matter of efficiency, simplicity, or security. In schemes targeting OAE1-security, the blocksize is usually small, like 128 bits, the value depending on the width of some underlying blockcipher or permutation used in the scheme's construction.

The conceptual problem with this design choice is, that the number of bits that are reasonable to buffer is application-environment specific. One application might need to limit the blocksize to 128 KB, so as to fit comfortably within the L2 cache of some CPU. Another application might need to limit the blocksize to 1 KB, to fit compactly on some ASIC or FPGA. Another application might need to limit the blocksize to a single byte, to ensure bounded latency despite bytes of plaintext arriving at unpredictable times.

The problem is that the designer of a cryptographic scheme is in no position to know the implementation-environment's constraints that would motivate the selection of a suitable blocksize. By choosing some fixed blocksize n, a scheme's designer simultaneously hinders an implementation's potential need to buffer less than n bits *and* an implementation's potential ability to buffer more than n bits of plaintext. *Any* choice of a blocksize replaces a user's environment-specific constraint by a hardwired choice from a primitive's designer.

Remark 6 (Blocksize vs. memory constraints)**.** *Before moving on let us point out that, if it is the amount of memory available to an implementation that is an issue, the right constraint is not the blocksize n, where block C_i depends only on prior blocks, but the requirement that an implementation be achievable in one pass and n bits of memory. These are not the same thing [RZ11, p. 241]. And the former is a poor substitute for the latter since context sizes vary substantially from scheme to scheme. While one could build an OAE notion with the amount of memory as an explicit parameter, we find it preferable to avoid such approach.*

Security must be defined for all plaintexts. The original OAE1[n] notion only defines security when messages are a multiple of n bits. Yet, such a message space is far from practical in a vast majority of AE applications, which work with a smaller, or simply different, data granularity than a typical value of n (e.g., $n = 128$, or perhaps $n = 64$ in legacy applications). In general, we think that an online-AE definition is not really meaningful (in practice) until one has specified what security means on the message space $\mathcal{M} = \{0,1\}^*$.

We note that saying "we pad first, there is no need to deal with strings that aren't multiples of the blocksize" does not really solve the raised issue, as it still leaves unspecified what is the *goal* one is aiming to achieve for the "incomplete" messages by applying the padding.[4]

Decryption too must be online. If one is able to produce ciphertext blocks in an online fashion one had better be able to decrypt them in the same fashion as they arrive. Perhaps the message was too long to be stored on the encrypting device. Then the same will likely hold on the decrypting device. Or perhaps there are timeliness constraints due to which one needs to act on a message fragment *now*, before the remainder of it arrives. For example, a video streaming service such as Netflix or YouTube needs to accommodate for such a constraint; it would be pointless to encrypt a video in an online fashion only to have to buffer the entire thing at the receiver's side before it could play.

But online decryption is not required by OAE1 security, and it is routine that online decryption of each provided block would be fatal. We conjecture that it is an unusual scenario where it is only important for encryption be computable online.

The OAE1 reference object is not ideal. The reference object for OAE1[n] security pre-supposes that encryption resembles an online-cipher followed by a random-looking tag. But it is wrong to think of this as capturing ideal behavior.

First, it implicitly assumes that all authenticity is taken care of at the very end. But if a plaintext is long and one is interested in encryption being online to ensure timeliness, then waiting until the entire ciphertext arrives to check authenticity makes no sense.

[4]There are natural ways to try to extend OAE1[n] security to a larger message space; see, for example, the approach used for online ciphers on $\{0,1\}^{\geq n}$ [RZ11]. This can be extended to OAE1. But it is not the only approach, and there will still be issues for dealing with strings of fewer than n bits.

Then, if one is going to act on a prefix of a plaintext as soon as it is recovered, it better be authenticated.

Second, it is simply irrelevant, from a security point of view, if, prior to receipt of an authentication tag, encryption amounts to length-preserving permutation. Doing this may minimize ciphertext length, but that is an efficiency issue, not a basic goal. And achieving this particular form of efficiency is at odds with possible authenticity aims.

7.5 Reformalizing Online AE

We remodel the online-AE and reformalize its security in a new notion. We will call it OAE2. To accurately model the underlying goal, not only must the security definition depart from (the games used in) NAE and MRAE, but so too must a scheme's basic syntax. In particular, we adopt an API-motivated view in which the segmentation of a plaintext is determined by the caller.

After defining the syntax we offer three ways to quantify the advantage an adversary gets in attacking an OAE2 scheme. We term these advantage measures OAE2a, OAE2b, OAE2c. The notions are essentially equivalent. We provide quantitative results to make this *essentially* precise.

We define the quantification of OAE2 security in three different advantage measures mainly to clarify what OAE2 really *is*. The measures have different characteristics. The first, OAE2a, is a vector-oriented formulation. It compares a scheme to a fairly easy-to-understand reference object. The second advantage measure, OAE2b, is a string-oriented formulation. It uses a tighter and more realistic accounting of the adversarial resources. The third advantage measure, OAE2c, is also string-oriented and is more aspirational in character. Yet it is the easiest notion to work with, at least for proving schemes OAE2-secure. The OAE2c measure only makes sense, however, if the segment-expansion τ is fairly large.

Segmented strings. In this chapter, we use the term *segmented-strings* to denote the vectors (or lists) of strings. Thus we call $\{0,1\}^{**} = (\{0,1\}^*)^*$ the set of segmented-strings, a segmented string $\boldsymbol{X} \in \{0,1\}^{**}$ is a vector of strings, and the segmented-string with zero components is the empty list Λ. Note that Λ is not the same as the empty string ε. We call the strings that are components of a segmented string segments. We refer the reader to the Section 2.1 for further notation used for vectors. We emphasize that a segmented string is not a string (and that an empty string and an empty segmented string are not the same thing, i.e. $\Lambda \neq \varepsilon$).

Online AE syntax. A *segmented-AE scheme* is a tuple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ where the *key space* $\mathcal{K}$ is a nonempty set with an associated distribution and both encryption $\mathcal{E} = (\mathcal{E}.\text{init}, \mathcal{E}.\text{next}, \mathcal{E}.\text{last})$ and decryption $\mathcal{D} = (\mathcal{D}.\text{init}, \mathcal{D}.\text{next}, \mathcal{D}.\text{last})$ are specified by triples of deterministic algorithms. Associated to Π are its *nonce space* $\mathcal{N} \subseteq \{0,1\}^*$ and its *state space* $\mathcal{S}$. For simplicity, a scheme's *AD space* $\mathcal{A} = \{0,1\}^*$, *message space*

```
algorithm $\boldsymbol{\mathcal{E}}(K, N, \boldsymbol{A}, \boldsymbol{M})$
$m \leftarrow |\boldsymbol{M}|$;  if $m = 0$ or $|\boldsymbol{A}| \neq |\boldsymbol{M}|$
then return $\Lambda$
$(A_1, \ldots, A_m) \leftarrow \boldsymbol{A}$
$(M_1, \ldots, M_m) \leftarrow \boldsymbol{M}$
$S_0 \leftarrow \mathcal{E}.\text{init}(K, N)$
for $i \leftarrow 1$ to $m - 1$ do
  $(C_i, S_i) \leftarrow \mathcal{E}.\text{next}(S_{i-1}, A_i, M_i)$
$C_m \leftarrow \mathcal{E}.\text{last}(S_{m-1}, A_m, M_m)$
return $(C_1, \ldots, C_m)$
```

```
algorithm $\boldsymbol{\mathcal{D}}(K, N, \boldsymbol{A}, \boldsymbol{C})$
$m \leftarrow |\boldsymbol{C}|$
if $m = 0$ or $|\boldsymbol{A}| \neq |\boldsymbol{C}|$ then return $\Lambda$
$(A_1, \ldots, A_m) \leftarrow \boldsymbol{A}$;  $(C_1, \ldots, C_m) \leftarrow \boldsymbol{C}$
$S_0 \leftarrow \mathcal{D}.\text{init}(K, N)$
for $i \leftarrow 1$ to $m - 1$ do
  if $\mathcal{D}.\text{next}(S_{i-1}, A_i, C_i) = \perp$ then
    if $m = 1$ return $\Lambda$
    else return $(M_1, \ldots, M_{i-1})$
  else $(M_i, S_i) \leftarrow \mathcal{D}.\text{next}(S_{i-1}, A_i, C_i)$
$M_m \leftarrow \mathcal{D}.\text{last}(S_{m-1}, A_m, C_m)$
if $M_m = \perp$ then
  return $(M_1, \ldots, M_{m-1})$
else return $(M_1, \ldots, M_m)$
```

Figure 7.5 – **Operating on segmented strings.** The figure shows the algorithms $\boldsymbol{\mathcal{E}}$ and $\boldsymbol{\mathcal{D}}$ that are *induced* by the segmented encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$.

$\mathcal{M} = \{0,1\}^*$, and *ciphertext space* $\mathcal{C} = \{0,1\}^*$ are all strings. While an AD will be provided with each plaintext segment, a single nonce is provided for the entire sequence of segments. The signatures of the components of $\mathcal{E}$ and $\mathcal{D}$ are as follows:

$\mathcal{E}.\text{init}\colon \mathcal{K} \times \mathcal{N} \to \mathcal{S}$	$\mathcal{D}.\text{init}\colon \mathcal{K} \times \mathcal{N} \to \mathcal{S}$
$\mathcal{E}.\text{next}\colon \mathcal{S} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C} \times \mathcal{S}$	$\mathcal{D}.\text{next}\colon \mathcal{S} \times \mathcal{A} \times \mathcal{C} \to (\mathcal{M} \times \mathcal{S}) \cup \{\perp\}$
$\mathcal{E}.\text{last}\colon \mathcal{S} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C}$	$\mathcal{D}.\text{last}\colon \mathcal{S} \times \mathcal{A} \times \mathcal{C} \to \mathcal{M} \cup \{\perp\}$

When an algorithm takes or produces a point $S \in \mathcal{S}$ from its state space, it is understood that a fixed encoding of S is employed.

Given a segmented-AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ there are *induced* encryption and decryption algorithms

$$\boldsymbol{\mathcal{E}}\colon \mathcal{K} \times \mathcal{N} \times \{0,1\}^{**} \times \{0,1\}^{**} \to \{0,1\}^{**} \text{ and } \boldsymbol{\mathcal{D}}\colon \mathcal{K} \times \mathcal{N} \times \{0,1\}^{**} \times \{0,1\}^{**} \to \{0,1\}^{**}$$

(note the change to bold font) that operate, all at once, on vectors of plaintext, ciphertext, and AD. These algorithms are defined in Figure 7.5. Observe how $\text{Dec}(K, N, \boldsymbol{A}, \boldsymbol{C})$ returns a longest $\boldsymbol{M}$ whose encryption (using K, N, and $\boldsymbol{A}$) is a prefix of $\boldsymbol{C}$; in essence, we stop at the first decryption failure, so $|\boldsymbol{C}| = |\boldsymbol{M}|$ if and only if $\boldsymbol{C}$ is entirely valid.

We require the following validity (or else correctness) condition for any segmented-AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with induced $(\boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{D}})$: if $K \in \mathcal{K}$, $N \in \mathcal{N}$, $\boldsymbol{A} \in \{0,1\}^{**}$, $\boldsymbol{M} \in \{0,1\}^{**}$, and $\boldsymbol{C} = \boldsymbol{\mathcal{E}}(K, N, \boldsymbol{A}, \boldsymbol{M})$, then $\boldsymbol{M} = \boldsymbol{\mathcal{D}}(K, N, \boldsymbol{A}, \boldsymbol{C})$.

Ciphertext expansion. We focus on segmented-AE schemes with constant segment-expansion, defined as follows: associated to Π is a number $\tau \geq 0$ such that if $K \in \mathcal{K}$, $N \in \mathcal{N}$, $\boldsymbol{A} \in \{0,1\}^{**}$, $\boldsymbol{M} \in \{0,1\}^{**}$, $m = |\boldsymbol{A}| = |\boldsymbol{M}|$, and $\boldsymbol{C} = \boldsymbol{\mathcal{E}}(K, N, \boldsymbol{A}, \boldsymbol{M})$, then $|\boldsymbol{C}[i]| = |\boldsymbol{M}[i]| + \tau$ for all $i \in [1..m]$. Thus each segment grows by exactly τ bits, for some constant τ. We call τ the *segment-expansion* of Π.

We favor constant segment-expansion because we think that the same level of authenticity ought to be guaranteed for the interior segments and for the final segment. After all, much of the point of online-AE is to allow a decrypting party to safely act on a ciphertext segment as soon as it is available. Still, there is an obvious price (in terms of efficiency) to pay for expanding every segment. See the paragraph "Multivalued segment-expansion" for the case where the amount of segment-expansion is position dependent.

Online computability. We say that a segmented-AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ has *online-encryption* if its state space $\mathcal{S}$ is finite and there is a constant w such that $\mathcal{E}$.next and $\mathcal{E}$.last use at most w bits of working memory. The value w excludes memory used for storing an algorithm's inputs or output; we elaborate below.

Similarly, scheme Π has *online-decryption* if its state space $\mathcal{S}$ is finite and there's a constant w' such that $\mathcal{D}$.next and $\mathcal{D}$.last use at most w' bits of working memory. A segmented-AE scheme is *online* if it has online-encryption and online-decryption.

In accounting for memory above, we assume a model of computation in which the input values are provided on a read-only input tape; the input's length is not a part of the working memory accounted for by w. Similarly, algorithms produce outputs by writing to a write-only output tape in a left-to-right fashion. The number of bits written out has nothing to do with the working memory w.

Our security definitions do not care if a segmented-AE scheme is online: that is an efficiency requirement, not a security requirement. Yet a good part of the purpose of the segmented-AE syntax is to properly model schemes that aim to comply with such efficiency constraints.

7.5.1 First OAE2 Definition: OAE2a

We begin by defining the *ideal* behavior for an OAE2 scheme. For any $\tau \in \mathbb{N}$, we endow the set of τ-expanding injections $\mathrm{Inj}(\tau)$ with a "uniform" distribution: sampling a random τ-expanding injection $f \leftarrow\$ \mathrm{Inj}(\tau)$ is equivalent with sampling $f_m \leftarrow\$ \mathrm{Inj}(2^m, 2^{m+\tau})$ for every $m \in \mathbb{N}$ and letting $f(M) = f_{|M|}(M)$ for all $M \in \{0,1\}^*$. We define a distribution on functions $F \leftarrow\$ \mathrm{IdealOAE}(\tau)$ as follows:

```
for $m \in \mathbb{N}$, $N \in \{0,1\}^*$, $\boldsymbol{A} \in (\{0,1\}^*)^m$, $\boldsymbol{M} \in (\{0,1\}^*)^{m-1}$ do
    $f_{N,\boldsymbol{A},\boldsymbol{M},0} \leftarrow\$ \mathrm{Inj}(\tau)$;   $f_{N,\boldsymbol{A},\boldsymbol{M},1} \leftarrow\$ \mathrm{Inj}(\tau)$
for $m \in \mathbb{N}$, $\boldsymbol{A} \in (\{0,1\}^*)^m$, $\boldsymbol{X} \in (\{0,1\}^*)^m$, $\delta \in \{0,1\}$ do
    $F(N, \boldsymbol{A}, \boldsymbol{X}, \delta) \leftarrow (f_{N,\boldsymbol{A}[1..1],\Lambda,0}(\boldsymbol{X}[1]),\ f_{N,\boldsymbol{A}[1..2],\boldsymbol{X}[1..1],0}(\boldsymbol{X}[2]),$
                $f_{N,\boldsymbol{A}[1..3],\boldsymbol{X}[1..2],0}(\boldsymbol{X}[3]),\ \ldots,\ f_{N,\boldsymbol{A}[1..m-1],\boldsymbol{X}[1..m-2],0}(\boldsymbol{X}[m-1]),$
                $f_{N,\boldsymbol{A}[1..m],\boldsymbol{X}[1..m-1],\delta}(\boldsymbol{X}[m]))$
return $F$
```

Thus $F \leftarrow\$ \mathrm{IdealOAE}(\tau)$ grows by accretion, the i^{th} component of $F(N, \boldsymbol{A}, \boldsymbol{X}, 0)$ de-

```
proc initialize                                   oae2a-R_Π
$K \leftarrow\$ \mathcal{K}$

oracle Enc($N, \boldsymbol{A}, \boldsymbol{M}$)
if $N \notin \mathcal{N}$ or $|\boldsymbol{A}| \neq |\boldsymbol{M}|$ then
    return $\bot$
return $\boldsymbol{\mathcal{E}}(K, N, \boldsymbol{A}, \boldsymbol{M})$

oracle Dec($N, \boldsymbol{A}, \boldsymbol{C}$)
if $N \notin \mathcal{N}$ or $|\boldsymbol{A}| \neq |\boldsymbol{M}|$ then
    return $\bot$
return $\boldsymbol{\mathcal{D}}(K, N, \boldsymbol{A}, \boldsymbol{C})$
```

```
proc initialize                                   oae2a-I_Π
$F \leftarrow\$$ IdealOAE($\tau$)

oracle Enc($N, \boldsymbol{A}, \boldsymbol{M}$)
if $N \notin \mathcal{N}$ or $|\boldsymbol{A}| \neq |\boldsymbol{M}|$ then
    return $\bot$
return $F(N, \boldsymbol{A}, \boldsymbol{M}, 1)$

oracle Dec($N, \boldsymbol{A}, \boldsymbol{C}$)
if $N \notin \mathcal{N}$ or $|\boldsymbol{A}| \neq |\boldsymbol{C}|$ then
    return $\bot$
if $\exists \boldsymbol{M}$ s.t. $F(N, \boldsymbol{A}, \boldsymbol{M}, 1) = \boldsymbol{C}$ then
    return $\boldsymbol{M}$
$\boldsymbol{M} \leftarrow$ LongestValidSubvector($F, N, \boldsymbol{A}, \boldsymbol{C}$)
return $\boldsymbol{M}$
```

Figure 7.6 – **OAE2a security.** The segmented-AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ has nonce space $\mathcal{N}$ and segment-expansion τ. It induces algorithms $\boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{D}}$ as per Figure 7.5. The notation LongestValidSubvector$(F, N, \boldsymbol{A}, \boldsymbol{C})$ stands for the longest vector in $\{\boldsymbol{M} : F(N, \boldsymbol{A}, \boldsymbol{M}, 0)[i] = \boldsymbol{C}[i] \text{ for } i \in [1..|\boldsymbol{M}|-1]\}$. The distribution IdealOAE(τ) is described in Section 7.5.1.

pending on N, $\boldsymbol{A}[1..i]$, and $\boldsymbol{X}[1..i]$. It must be decryptable (hence the injectivity) and have the mandated length. The final input to F, the flag δ, indicates if the argument $\boldsymbol{X}$ is complete: a 1 means it is, a 0 means it's not. Figure 7.6 defines games **oae2a-R**$_\Pi$ and **oae2a-I**$_\Pi$ for a τ-expanding segmented-AE scheme Π. We note that a sampled function $F \leftarrow\$$ IdealOAE(τ) can be efficiently implemented by lazy sampling.[5]

Definition 7.2 (OAE2a security)**.** *Given a segmented-AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with ciphertext expansion $\tau \in \mathbb{N}$, and given an adversary $\mathscr{A}$, we define the advantage of $\mathscr{A}$ in breaking the* OAE2a *AE security of* Π *in a chosen ciphertext attack (with help of the games* **oae2a-*R*** *and* **oae2a-*I*** *in Figure 7.6) as*

$$\mathbf{Adv}_\Pi^{\mathbf{oae2a}}(\mathscr{A}) = \Pr[\mathscr{A}^{\mathbf{oae2a}\text{-}\boldsymbol{R}_\Pi} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{oae2a}\text{-}\boldsymbol{I}_\Pi} \Rightarrow 1].$$

If $\mathbf{Adv}_\Pi^{\mathbf{oae2a}}(\mathscr{A}) \leq \epsilon$ for all adversaries $\mathscr{A}$ whose running time is limited by t, whose queries contain no more than q segments in total, and whose data complexity (in bits) in all queries is limited by σ then we say that Π is a (ϵ, t, q, σ)-secure OAE2a *scheme.*

Discussion. We now explain the intuition behind the notion OAE2a. A user of online AE wants to encrypt a segmented message $\boldsymbol{M} = (M_1, \ldots, M_m)$ into a ciphertext $\boldsymbol{C} = (C_1, \ldots, C_m)$ using $K, N, \boldsymbol{A}$. He/she wants to do this *as well as possible* subject to

[5]A random τ-expanding injection can be implemented similarly as a random online permutation. A function $F \leftarrow\$$ IdealOAE(τ) can then be implemented as a dynamically extendible collection of random injections.

the constraint that segments grow by exactly τ bits and $M_1 \cdots M_i$ are recoverable from $K, N, (A_1, \ldots, A_i), (C_1, \ldots, C_i)$ (to allow online decryption).

Similarly as the security notion of robust-AE [HKR15], we target an achievable (instead of aspirational) goal, which is signalled by the phrase "as well as possible" . Specifically, our goal is formalized by comparing a real scheme to a random element from IdealOAE(τ) and its inverse, the latter understood to invert as many components as possible, stopping at the first point one can't proceed.

The definition of IdealOAE(τ) is complex enough that an example may help. Consider encrypting a segmented plaintext $\boldsymbol{M} = (A, B, C, D)$ with a fixed key, nonce, and AD. Let (U, V, X, Y) be the result. Now encrypt $\boldsymbol{M'} = (A, B, C)$. We want the encryption algorithm to return (U, V, Z), not (U, V, X), as the final segment is special: processed by $\mathcal{E}$.last instead of $\mathcal{E}$.next, it is as though $\boldsymbol{M} = (A, B, C, D)$ means $(A, B, C, D\$)$, while $\boldsymbol{M} = (A, B, C)$ means $(A, B, C\$)$, where the \$-symbol is an end-of-message sentinel. Written like this, it is clear that the two segmented ciphertexts should agree on the first two components but not the third. Correspondingly, possession of (U, V, X, Y) ought not enable a forgery of (U, V, X). All of this understanding gets quietly embedded into the definition of IdealOAE(τ), whose member functions get a final argument δ with semantics indicating if the message is *complete*. Thus $F(N, \boldsymbol{A}, (A, B, C), 0)$ is what $\boldsymbol{M} = (A, B, C)$ should map to if more segments are to come, while $F(N, \boldsymbol{A}, (A, B, C), 1)$ is what it should map to if C is the final segment of $\boldsymbol{M}$.

7.5.2 Second OAE2 Definition: OAE2b

The games in Figure 7.7 provide a more fine-grained and string-oriented measure for OAE2 security. The adversary, instead of querying $N, \boldsymbol{A}, \boldsymbol{M}$ and getting a vector $\boldsymbol{C} = \text{Enc}(N, \boldsymbol{A}, \boldsymbol{M})$, can adaptively grow $\boldsymbol{A}$ and $\boldsymbol{M}$ one component at a time. Similarly, instead of providing a segmented ciphertext $N, \boldsymbol{A}, \boldsymbol{C}$ and getting $\boldsymbol{M} = \text{Dec}(N, \boldsymbol{A}, \boldsymbol{C})$, it can adaptively grow $\boldsymbol{A}, \boldsymbol{C}$.

Definition 7.3 (OAE2b security). *Given a segmented-AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with ciphertext expansion $\tau \in \mathbb{N}$, and given an adversary $\mathscr{A}$, we define the advantage of $\mathscr{A}$ in breaking the* OAE2b *AE security of* Π *in a chosen ciphertext attack (with help of the games* **oae2b-*R*** *and* **oae2b-*I*** *in Figure 7.7) as*

$$\mathbf{Adv}_{\Pi}^{\mathbf{oae2b}}(\mathscr{A}) = \Pr[\mathscr{A}^{\mathbf{oae2b}\text{-}\boldsymbol{R}_{\Pi}} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{oae2b}\text{-}\boldsymbol{I}_{\Pi}} \Rightarrow 1].$$

If $\mathbf{Adv}_{\Pi}^{\mathbf{oae2b}}(\mathscr{A}) \le \epsilon$ for all adversaries $\mathscr{A}$ whose running time is limited by t, who make no more than q calls to any of the interfaces of both encryption and decryption, and whose data complexity (in bits) in all queries is limited by σ then we say that Π *is a (ϵ, t, q, σ)-secure* OAE2b *scheme.*

The OAE2a and OAE2b measures are essentially equivalent. The meaning of "essentially" is made precise by a simple result explaining how to convert an adversary for one definition into an adversary for the other. This is stated formally in Proposition 7.4.

```
proc initialize                    $\boxed{\textbf{oae2b-R}_{\Pi}}$
$I, J \leftarrow 0$;  $K \leftarrow\!\!\$\; \mathcal{K}$

oracle Enc.init($N$)
if $N \notin \mathcal{N}$ then
    return $\bot$
$I \leftarrow I + 1$;  $S_I \leftarrow \mathcal{E}.\mathrm{init}(K, N)$
return $I$

oracle Enc.next($i, A, M$)
if $i \notin [1..I]$ or $S_i = \bot$ then
    return $\bot$
$(C, S_i) \leftarrow \mathcal{E}.\mathrm{next}(S_i, A, M)$
return $C$

oracle Enc.last($i, A, M$)
if $i \notin [1..I]$ or $S_i = \bot$ then
    return $\bot$
$C \leftarrow \mathcal{E}.\mathrm{last}(S_i, A, M)$
$S_i \leftarrow \bot$
return $C$

oracle Dec.init($N$)
if $N \notin \mathcal{N}$ then
    return $\bot$
$J \leftarrow J + 1$;  $S'_J \leftarrow \mathcal{D}.\mathrm{init}(K, N)$
return $J$

oracle Dec.next($j, A, C$)
if $j \notin [1..J]$ or $S'_j = \bot$ then
    return $\bot$
$(M, S'_j) \leftarrow \mathcal{D}.\mathrm{next}(S'_j, A, C)$
return $M$

oracle Dec.last($j, A, C$)
if $j \notin [1..J]$ or $S'_j = \bot$ then
    return $\bot$
$M \leftarrow \mathcal{D}.\mathrm{last}(S'_j, A, C)$
$S'_j \leftarrow \bot$
return $M$
```

```
proc initialize                    $\boxed{\textbf{oae2b-I}_{\Pi}}$
$I, J \leftarrow 0$;  $F \leftarrow\!\!\$\; \mathrm{IdealOAE}(\tau)$

oracle Enc.init($N$)
if $N \notin \mathcal{N}$ then return $\bot$
$I \leftarrow I + 1$;  $N_I \leftarrow N$
$A_I \leftarrow \Lambda$;  $M_I \leftarrow \Lambda$
return $I$

oracle Enc.next($i, A, M$)
if $i \notin [1..I]$ or $\boldsymbol{M}_i = \bot$ then return $\bot$
$\boldsymbol{A}_i \leftarrow \boldsymbol{A}_i \| A$;  $\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i \| M$
$m \leftarrow |\boldsymbol{M}_i|$;  $\boldsymbol{C} \leftarrow F(N_i, \boldsymbol{A}_i, \boldsymbol{M}_i, 0)$
return $\boldsymbol{C}[m]$

oracle Enc.last($i, A, M$)
if $i \notin [1..I]$ or $\boldsymbol{M}_i = \bot$ then return $\bot$
$\boldsymbol{A}_i \leftarrow \boldsymbol{A}_i \| A$;  $\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i \| M$
$m \leftarrow |\boldsymbol{M}_i|$;  $\boldsymbol{C} \leftarrow F(N_i, \boldsymbol{A}_i, \boldsymbol{M}_i, 1)$
$\boldsymbol{M}_i \leftarrow \bot$
return $\boldsymbol{C}[m]$

oracle Dec.init($N$)
if $N \notin \mathcal{N}$ then return $\bot$
$J \leftarrow J + 1$;  $N'_J \leftarrow N$
$\boldsymbol{A}'_j \leftarrow\!\!\$\; \Lambda$;  $\boldsymbol{C}_J \leftarrow \Lambda$
return $J$

oracle Dec.next($j, A, C$)
if $j \notin [1..J]$ or $\boldsymbol{C}_j = \bot$ then return $\bot$
$\boldsymbol{A}'_j \leftarrow \boldsymbol{A}_j \| A$;  $\boldsymbol{C}_j \leftarrow \boldsymbol{C}_j \| C$;  $m \leftarrow |\boldsymbol{C}_j|$
if $\exists \boldsymbol{M}$ s.t. $F(N'_j, \boldsymbol{A}'_j, \boldsymbol{M}, 0) = \boldsymbol{C}_j$ then
    return $\boldsymbol{M}[m]$
else
    $\boldsymbol{C}_j \leftarrow \bot$;  return $\bot$

oracle Dec.last($j, A, C$)
if $j \notin [1..J]$ or $\boldsymbol{C}_j = \bot$ then return $\bot$
$\boldsymbol{A}'_j \leftarrow \boldsymbol{A} \| A$;  $\boldsymbol{C}_j \leftarrow \boldsymbol{C}_j \| C$;  $m \leftarrow |\boldsymbol{C}_j|$
if $\exists \boldsymbol{M}$ s.t. $F(N'_j, \boldsymbol{A}'_j, \boldsymbol{M}_j, 1) = \boldsymbol{C}_j$ then
    $\boldsymbol{C}_j \leftarrow \bot$;  return $\boldsymbol{M}[m]$
else
    $\boldsymbol{C}_j \leftarrow \bot$;  return $\bot$
```

Figure 7.7 – **OAE2b security.** The segmented-AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ has nonce space $\mathcal{N}$ and segment-expansion τ. The distribution IdealOAE(τ) is defined in Section 7.5.1.

Proposition 7.4 (OAE2a $\approx$ OAE2b). *Let Π be a segmented-AE scheme with ciphertext expansion τ. For any* OAE2a *adversary $\mathscr{A}_1$ hat runs in time t_1 and queries σ_1 segments in total, there is an adversary $\mathscr{B}_1$ that has*

$$\mathbf{Adv}_{\Pi}^{\mathbf{oae2a}}(\mathscr{A}_1) \leq \mathbf{Adv}_{\Pi}^{\mathbf{oae2b}}(\mathscr{B}_1)$$

such that $\mathscr{B}_1$ runs in time $t_1 + \gamma_1 \cdot \sigma_1$ for some constant γ_1, and queries the same number of segments in total as does $\mathscr{A}_1$.

For any OAE2b *adversary $\mathscr{B}_2$ that runs in time t_2 and queries up to σ_2 segments in total, there is an adversary $\mathscr{A}_2$ that has*

$$\mathbf{Adv}_{\Pi}^{\mathbf{oae2b}}(\mathscr{B}_2) \leq \mathbf{Adv}_{\Pi}^{\mathbf{oae2a}}(\mathscr{A}_2)$$

such that $\mathscr{A}_2$ runs in time $t_2 + \gamma_2 \cdot \sigma_2^2$ for some constant γ_2, and queries up to σ_2^2 segments in total.

Proof. First, given an OAE2a adversary $\mathscr{A}_1$ we construct an equally effective OAE2b adversary $\mathscr{B}_1$: it translates each query $\text{Enc}(N, (A_1, \ldots, A_m), (M_1, \ldots, M_m))$ asked by adversary $\mathscr{A}_1$ into an Enc.init, then $m-1$ Enc.next calls, then an Enc.last call, assembling the answers into a segmented ciphertext $(C_1, \ldots, C_m)$. Similarly, it translates any query $\text{Dec}(N, (A_1, \ldots, A_m), (C_1, \ldots, C_m))$ into a sequence of Dec.init, Dec.next, Dec.last calls. Adversary $\mathscr{B}_1$ gets exactly the OAE2b-advantage that $\mathscr{A}_1$ had as OAE2a-advantage. It runs in almost the exact same time.

Simulation in the other direction is less efficient. Given an adversary $\mathscr{B}_2$ attacking the OAE2b-security of a Π, we construct an adversary $\mathscr{A}_2$ for attacking the OAE2a-security of the same scheme. Adversary $\mathscr{A}_2$ maintains lists $N_i, \boldsymbol{A}_i, \boldsymbol{M}_i$ that are initialized in the natural way with each Enc.init call (incrementing i, initially zero, with each Enc.init). Calls of the form $\text{Enc.next}(i, A, M)$, when valid, result in appending A to $\boldsymbol{A}_i$ and M to $\boldsymbol{M}_i$, making an $\text{Enc}(N_i, \boldsymbol{A}_i \| \varepsilon, \boldsymbol{M}_i \| \varepsilon)$ call, and returning its $|\boldsymbol{M}_i|$-th component. Calls of the form $\text{Enc.last}(i, A, M)$ result in making an $\text{Enc}(N_i, \boldsymbol{A}_i \| A, \boldsymbol{M}_i \| M)$ call, returning its last component, resetting $\boldsymbol{M}_i$ to $\perp$ before doing so. Calls of the form Dec.init, Dec.next, and Dec.last are treated analogously, maintaining $N_i', A_i', \boldsymbol{C}_i$ values. Once again the simulation is perfect, so $\mathbf{Adv}_{\Pi}^{\mathbf{oae2a}}(\mathscr{A}_2) = \mathbf{Adv}_{\Pi}^{\mathbf{oae2b}}(\mathscr{B}_2)$. But now there is a quadratic slowdown in running time: the argument lists can grow long, as can return values, only one component of which is used with each call. $\square$

While the OAE2a definition is more compact, the adversary's ability to ask segmentwise adaptive queries in the OAE2b definition ultimately makes it preferable, particularly as this better models the real-world semantics; an adversary might be able to incrementally grow a plaintext or ciphertext with the unwitting cooperation of some encrypting or decrypting party. [6]

[6] We note that we could grant the adversary with even greater ability to adapt its queries that would allow it to grow a tree, and not just a chain of segments. But this would not seem to model anything meaningful in the real-world.

There are a couple of further reasons to favor OAE2b. One is that it more directly captures the possibility of "infinite" (non-terminating) plaintexts (an infinite "stream" of messages). This is simply the setting where Enc.last and Dec.last are never called. Second, the OAE2b definition makes it easier to define nonce-respecting adversaries for the OAE setting. Such adversaries may adaptively grow a plaintext based on a single nonce, but it may grow only *one* plaintext for any given nonce. Building on the OAE2a formulation this is awkward to say, but building on the OAE2b formulation, it is natural.

7.5.3 Third OAE2 Definition: OAE2c

Let Π be a segmented-AE scheme with segment-expansion τ and nonce-space $\mathcal{N}$. Our final formulation of OAE2 security uses a two-part definition, separately defining privacy and authenticity requirements with help of the games defined in Figure 7.8.

Definition 7.5 (OAE2c security)**.** *Given a segmented-AE scheme* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *with ciphertext expansion* $\tau \in \mathbb{N}$*, and given an adversary* $\mathscr{A}$*, we define the advantage of* $\mathscr{A}$ *in breaking the confidentiality of* Π *in a chosen plaintext attack (with help of the games* **oae2c-*R*** *and* **oae2c-*I*** *in Figure 7.8) as*

$$\mathbf{Adv}_{\Pi}^{\mathbf{oae2\text{-}priv}}(\mathscr{A}) = \Pr[\mathscr{A}^{\mathbf{oae2c\text{-}}\boldsymbol{R}_\Pi} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{oae2c\text{-}}\boldsymbol{I}_\Pi} \Rightarrow 1].$$

Given an adversary $\mathscr{A}'$*, we define the advantage of* $\mathscr{A}'$ *in breaking the authenticity of* Π *in a chosen ciphertext attack (with help of the game* **oae2c-*F*** *in Figure 7.8) as*

$$\mathbf{Adv}_{\Pi}^{\mathbf{oae2\text{-}auth}}(\mathscr{A}') = \Pr[\mathscr{A}^{\mathbf{oae2c\text{-}}\boldsymbol{F}_\Pi} \Rightarrow \mathsf{true}]$$

where $\mathscr{A}^{\mathbf{oae2c\text{-}}\boldsymbol{F}_\Pi} \Rightarrow \mathsf{true}$ *denotes the event that* $\mathscr{A}'$ *returns a value that, when provided as input to the procedure* ***finalize****, evaluates to* true*.*

If $\mathbf{Adv}_{\Pi}^{\mathbf{oae2\text{-}priv}}(\mathscr{A}) \leq \epsilon$ *for all adversaries* $\mathscr{A}$ *whose running time is limited by* t*, who make no more than* q *calls to any of the interfaces of both encryption and decryption, and whose data complexity (in bits) in all queries is limited by* σ *then we say that* Π *is a* (ϵ, t, q, σ)*-privacy-secure* OAE2c *scheme.*

If $\mathbf{Adv}_{\Pi}^{\mathbf{oae2\text{-}auth}}(\mathscr{A}') \leq \epsilon'$ *for all adversaries* $\mathscr{A}'$ *whose running time is limited by* t'*, who make no more than* q' *calls to any of the interfaces of both encryption and decryption, and whose data complexity (in bits) in all queries is limited by* σ' *then we say that* Π *is a* $(\epsilon', t', q', \sigma')$*-authenticity-secure* OAE2c *scheme.*

Definition OAE2c is simpler than the prior two in the sense that, for privacy, no decryption oracles are provided and the reference experiment simply returns the right number of uniformly random bits. For the authenticity portion of the definition, forgeries are defined to allow any $(N, \boldsymbol{A}, \boldsymbol{C})$ that the adversary does not trivially know to be valid, the adversary marking if $\boldsymbol{C}$ has terminated ($b = 1$) or not ($b = 0$). The set $\mathcal{Y}$ records the tuples that the adversary knows to be trivially correct from its encryption queries.

```
proc initialize $\boxed{\text{oae2c-R}_\Pi}$ $\boxed{\text{oae2c-F}_\Pi}$
$I \leftarrow 0$;  $K \leftarrow\!\!\$\ \mathcal{K}$
$\mathcal{Y} \leftarrow \emptyset$

oracle Enc.init$(N)$
if $N \notin \mathcal{N}$ then
        return $\perp$
$I \leftarrow I + 1$;  $S_I \leftarrow \mathcal{E}.\text{init}(K, N)$
$N_I \leftarrow N$;  $\boldsymbol{A}_I \leftarrow \boldsymbol{M}_I \leftarrow \boldsymbol{C}_I \leftarrow \Lambda$
return  $I$

oracle Enc.next$(i, A, M)$
if $i \notin [1..I]$ or $S_i = \perp$ then
        return $\perp$
$(C, S_i) \leftarrow \mathcal{E}.\text{next}(S_i, A, M)$
$\boldsymbol{A}_i \leftarrow \boldsymbol{A}_i \| A$;  $\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i \| M$
$\boldsymbol{C}_i \leftarrow \boldsymbol{C}_i \| C$;  $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(N_i, \boldsymbol{A}_i, \boldsymbol{C}_i, 0)\}$
return $C$

oracle Enc.last$(i, A, M)$
if $i \notin [1..I]$ or $S_i = \perp$ then
        return $\perp$
$C \leftarrow \mathcal{E}.\text{last}(S_i, A, M)$;  $S_i \leftarrow \perp$
$\boldsymbol{A}_i \leftarrow \boldsymbol{A}_i \| A$;  $\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i \| M$
$\boldsymbol{C}_i \leftarrow \boldsymbol{C}_i \| C$;  $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(N_i, \boldsymbol{A}_i, \boldsymbol{C}_i, 1)\}$
return $C$

◇proc finalize $(N, \boldsymbol{A}, \boldsymbol{C}, b)$
◇if $N \notin \mathcal{N}$ or $(N, \boldsymbol{A}, \boldsymbol{C}, b) \in \mathcal{Y}$ then
◇       return false
◇if $|\boldsymbol{A}| \neq |\boldsymbol{C}|$ or $|\boldsymbol{A}| = 0$ then
◇       return false
◇$S \leftarrow \mathcal{D}.\text{init}(K, N)$;  $m \leftarrow |\boldsymbol{C}|$
◇for $i \leftarrow 1$ to $m - b$ do
◇       $(M, S) \leftarrow \mathcal{D}.\text{next}(S, \boldsymbol{A}[i], \boldsymbol{C}[i])$
◇       if $M = \perp$ then return false
◇if $b = 1$ and $\mathcal{D}.\text{last}(S, \boldsymbol{A}[m], \boldsymbol{C}[m]) = \perp$
◇       then return false
◇return true
```

```
proc initialize                    $\boxed{\text{oae2c-I}_\Pi}$
$I \leftarrow 0$
$E(x) \leftarrow \perp$ for all $x$

oracle Enc.init$(N)$
if $N \notin \mathcal{N}$ then
        return $\perp$
$I \leftarrow I + 1$
$N_I \leftarrow N$;  $\boldsymbol{A}_i \leftarrow \boldsymbol{M}_i \leftarrow \Lambda$
return  $I$

oracle Enc.next$(i, A, M)$
if $i \notin [1..I]$ or $N_i = \perp$ then
        return $\perp$
$\boldsymbol{A}_i \leftarrow \boldsymbol{A}_i \| A$;  $\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i \| M$
if $E(N_i, \boldsymbol{A}_i, \boldsymbol{M}_i, 0) = \perp$ then
        $E(N_i, \boldsymbol{A}_i, \boldsymbol{M}_i, 0) \leftarrow\!\!\$\ \{0,1\}^{|M|+\tau}$
$C \leftarrow E(N_i, \boldsymbol{A}_i, \boldsymbol{M}_i, 0)$
return $C$

oracle Enc.last$(i, A, M)$
if $i \notin [1..I]$ or $N_i = \perp$ then
        return $\perp$
$\boldsymbol{A}_i \leftarrow \boldsymbol{A}_i \| A$;  $\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i \| M$
if $E(N_i, \boldsymbol{A}_i, \boldsymbol{M}_i, 1) = \perp$ then
        $E(N_i, \boldsymbol{A}_i, \boldsymbol{M}_i, 1) \leftarrow\!\!\$\ \{0,1\}^{|M|+\tau}$
$C \leftarrow E(N_i, \boldsymbol{A}_i, \boldsymbol{M}_i, 1)$;  $N_i \leftarrow \perp$
return $C$
```

Figure 7.8 – **OAE2c security.** Privacy and authenticity are separately defined, the first by comparing games **oae2c-R** and **oae2c-I**, and the second using game **oae2c-F**, which includes the additional lines indicated by ◇.

The following propositions show that OAE2b and OAE2c are close, assuming that the segment-expansion τ is fairly large.

Proposition 7.6 (OAE2c $\Rightarrow$ OAE2b)**.** *Let* Π *be a segmented-AE scheme with ciphertext expansion* τ*. For any adversary* $\mathscr{A}$ *that runs in time* t*, makes no more than* q *individual queries and starts no more than* p *decryption chains, such that it queries no more than* σ *bits of data in total, there are adversaries* $\mathscr{B}_1$ *and* $\mathscr{B}_2$ *for which*

$$\mathbf{Adv}_{\Pi}^{\mathbf{oae2b}}(\mathscr{A}) \le \mathbf{Adv}_{\Pi}^{\mathbf{oae2\text{-}priv}}(\mathscr{B}_1) + p \cdot \mathbf{Adv}_{\Pi}^{\mathbf{oae2\text{-}auth}}(\mathscr{B}_2) + q^2/2^{\tau}.$$

For each $i \in \{1,2\}$*, adversary* $\mathscr{B}_i$ *runs in time* $t+\gamma_i \cdot \sigma$ *for some constant* γ_i*, and queries at most* σ *bits in total.*

```
proc Enc.init($N$)
$I \leftarrow I+1$; $N_I \leftarrow N$; $S_I \leftarrow \varepsilon$
$\boldsymbol{M}_I, \boldsymbol{A}_I, \boldsymbol{C}_I \leftarrow \Lambda$
return Enc.$\overline{\text{init}}$($N$)

proc Enc.next($i, A, M$)
if $i > I$ or $S_i = \bot$ then return $\bot$
$C \leftarrow$ Enc.$\overline{\text{next}}$($i, A, M$)
$\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i \| M$; $\boldsymbol{A}_i \leftarrow \boldsymbol{A}_i \| A$
$\boldsymbol{C}_i \leftarrow \boldsymbol{C}_i \| C$; $H[N_i, \boldsymbol{A}_i, \boldsymbol{C}_i, 0] \leftarrow \boldsymbol{M}_i$
return $C$

proc Enc.last($i, A, M$)
if $i > I$ or $S_i = \bot$ then return $\bot$
$C \leftarrow$ Enc.$\overline{\text{last}}$($i, A, M$); $S_i \leftarrow \bot$
$\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i \| M$; $\boldsymbol{A}_i \leftarrow \boldsymbol{A}_i \| A$
$\boldsymbol{C}_i \leftarrow \boldsymbol{C}_i \| C$; $H[N_i, \boldsymbol{A}_i, \boldsymbol{C}_i, 1] \leftarrow \boldsymbol{M}_i$;
return $C$
```

```
proc Dec.init($N$)
$J \leftarrow J+1$; $N'_J \leftarrow N$; $S'_J \leftarrow \varepsilon$
$\boldsymbol{A}'_J, \boldsymbol{C}'_J \leftarrow \Lambda$
return $J$

proc Dec.next($j, A, C$)
if $j > J$ or $S'_j = \bot$ then return $\bot$
$\boldsymbol{A}'_j \leftarrow \boldsymbol{A}'_j \| A$; $\boldsymbol{C}'_j \leftarrow \boldsymbol{C}$
if $H[N'_j, \boldsymbol{A}'_j, \boldsymbol{C}'_j, 0] \ne \bot$ then
    $\boldsymbol{M} \leftarrow H[N'_j, \boldsymbol{A}'_j, \boldsymbol{C}'_j, 0]$
    return $\boldsymbol{M}[|\boldsymbol{M}|]$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(N'_j, \boldsymbol{A}'_j, \boldsymbol{C}'_j, 0)\}$; $S'_j \leftarrow \bot$;
return $\bot$

proc Dec.last($j, A, C$)
if $j > J$ or $S'_j = \bot$ then return $\bot$
$\boldsymbol{A}'_j \leftarrow \boldsymbol{A}'_j \| A$; $\boldsymbol{C}'_j \leftarrow \boldsymbol{C}$; $S'_j \leftarrow \bot$
if $H[N'_j, \boldsymbol{A}'_j, \boldsymbol{C}'_j, 0] \ne \bot$ then
    $\boldsymbol{M} \leftarrow H[N'_j, \boldsymbol{A}'_j, \boldsymbol{C}'_j, 1]$
    return $\boldsymbol{M}[|\boldsymbol{M}|]$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(N'_j, \boldsymbol{A}'_j, \boldsymbol{C}'_j, 1)\}$
return $\bot$
```

Figure 7.9 – **The procedures that $\mathscr{B}_i$ (with $i \in \{1,2\}$) runs in the proof of Proposition 7.6** to simulate $\mathscr{A}$'s oracles. The procedures Enc.$\overline{\text{init}}$, Enc.$\overline{\text{next}}$, Enc.$\overline{\text{last}}$ are the encryption oracles of $\mathscr{B}_i$. There is an implicit procedure **initialize**() that initializes $I, J \leftarrow 0$, $\mathcal{Y} \leftarrow \emptyset$ and H to an empty array.

Proof. We construct the adversary $\mathscr{B}_1$ from $\mathscr{A}$ as follows. The former runs the latter, implementing $\mathscr{A}$'s oracles as indicated in Figure 7.9, and outputs the same guess as $\mathscr{A}$. Next, we create adversary $\mathscr{B}_2$ from $\mathscr{A}$ as follows. The former runs the latter, implementing $\mathscr{A}$'s oracles as indicated in Figure 7.9. When $\mathscr{A}$ terminates, $\mathscr{B}_2$ will process the resulting set $\mathcal{Y}$. For $(N, \boldsymbol{A}, \boldsymbol{C}, 0)$ and $(N, \boldsymbol{A}', \boldsymbol{C}', \delta)$ in $\mathcal{Y}$, we'll delete the former vector if $m = |\boldsymbol{A}| < |\boldsymbol{A}'|$, and $\boldsymbol{A}[i] = \boldsymbol{A}'[i]$ and $\boldsymbol{C}[i] = \boldsymbol{C}'[i]$ for every $i \le m$. Now the set $\mathcal{Y}$ will

```
Games $G_1$, $\boxed{G_2}$

proc Enc.init($N$)
  $I \leftarrow I + 1$; $N_I \leftarrow N$
  $S_I \leftarrow \mathcal{E}.\text{init}(K, N)$; $\boldsymbol{M}_I, \boldsymbol{A}_I, \boldsymbol{C}_I \leftarrow \Lambda$
  return $I$

proc Enc.next($i, A, M$)
  if $i > I$ or $S_i = \perp$ then return $\perp$
  $C \leftarrow \mathcal{E}.\text{next}(S_i, A, M)$
  $\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i \| M$; $\boldsymbol{A}_i \leftarrow \boldsymbol{A}_i \| A$
  $\boldsymbol{C}_i \leftarrow \boldsymbol{C}_i \| C$; $H[N_i, \boldsymbol{A}_i, \boldsymbol{C}_i, 0] \leftarrow \boldsymbol{M}_i$;
  return $C$

proc Enc.last($i, A, M$)
  if $i > I$ or $S_i = \perp$ then return $\perp$
  $C \leftarrow \mathcal{E}.\text{last}(S_i, A, M)$; $S_i \leftarrow \perp$
  $\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i \| M$; $\boldsymbol{A}_i \leftarrow \boldsymbol{A}_i \| A$
  $\boldsymbol{C}_i \leftarrow \boldsymbol{C}_i \| C$; $H[N_i, \boldsymbol{A}_i, \boldsymbol{C}_i, 1] \leftarrow \boldsymbol{M}_i$;
  return $C$

proc Dec.init($N$)
  $J \leftarrow J + 1$; $N'_J \leftarrow N$
  $S'_J \leftarrow \mathcal{D}.\text{init}(K, N)$; $\boldsymbol{M}'_J, \boldsymbol{A}'_J, \boldsymbol{C}'_J \leftarrow \Lambda$
  return $J$

proc Dec.next($j, A, C$)
  if $j > J$ or $S'_j = \perp$ then return $\perp$
  $\boldsymbol{A}'_j \leftarrow \boldsymbol{A}'_j \| A$; $\boldsymbol{C}'_j \leftarrow \boldsymbol{C}'_j \| C$
  if $H[N'_j, \boldsymbol{A}'_j, \boldsymbol{C}'_j, 0] \neq \perp$ then
    $\boldsymbol{M} \leftarrow H[N'_j, \boldsymbol{A}'_j, \boldsymbol{C}'_j, 0]$
    return $\boldsymbol{M}[|\boldsymbol{M}|]$
  $(M, S'_j) \leftarrow \mathcal{D}.\text{next}(S'_j, A, C)$
  if $M \neq \perp$ then bad $\leftarrow$ true; $\boxed{S'_j, M \leftarrow \perp}$
  return $M$

proc Dec.last($j, A, C$)
  if $j > J$ or $S'_j = \perp$ then return $\perp$
  $\boldsymbol{A}'_j \leftarrow \boldsymbol{A}'_j \| A$; $\boldsymbol{C}'_j \leftarrow \boldsymbol{C}'_j \| C$
  if $H[N'_j, \boldsymbol{A}'_j, \boldsymbol{C}'_j, 0] \neq \perp$ then
    $\boldsymbol{M} \leftarrow H[N'_j, \boldsymbol{A}'_j, \boldsymbol{C}'_j, 1]$; $S'_j \leftarrow \perp$
    return $\boldsymbol{M}[|\boldsymbol{M}|]$
  if $M \neq \perp$ then bad $\leftarrow$ true; $\boxed{M \leftarrow \perp}$
  $S'_j \leftarrow \perp$;
  return $M$
```

```
Game $G_3$

proc Enc.init($N$)
  $I \leftarrow I + 1$; $N_I \leftarrow N$
  $S_I \leftarrow \varepsilon$; $\boldsymbol{M}_I, \boldsymbol{A}_I, \boldsymbol{C}_I \leftarrow \Lambda$
  return $I$

proc Enc.next($i, A, M$)
  if $i > I$ or $S_i = \perp$ then return $\perp$
  $\boldsymbol{A}_i \leftarrow \boldsymbol{A}_i \| A$; $C \leftarrow \rho_{N_i, \boldsymbol{A}_i, \boldsymbol{M}_i, 0}(M)$
  $\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i \| M$; $\boldsymbol{C}_i \leftarrow \boldsymbol{C}_i \| C$
  $H[N_i, \boldsymbol{A}_i, \boldsymbol{C}_i, 0] \leftarrow \boldsymbol{M}_i$
  return $C$

proc Enc.last($i, A, M$)
  if $i > I$ or $S_i = \perp$ then return $\perp$
  $\boldsymbol{A}_i \leftarrow \boldsymbol{A}_i \| A$; $C \leftarrow \rho_{N_i, \boldsymbol{A}_i, \boldsymbol{M}_i, 1}(M)$
  $S_i \leftarrow \perp$; $\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i \| M$
  $\boldsymbol{C}_i \leftarrow \boldsymbol{C}_i \| C$; $H[N_i, \boldsymbol{A}_i, \boldsymbol{C}_i, 1] \leftarrow \boldsymbol{M}_i$
  return $C$

proc Dec.init($N$)
  $J \leftarrow J + 1$; $N'_J \leftarrow N$
  $S'_J \leftarrow \varepsilon$; $\boldsymbol{M}'_J, \boldsymbol{A}'_J, \boldsymbol{C}'_J \leftarrow \Lambda$
  return $J$

proc Dec.next($j, A, C$)
  if $j > J$ or $S'_j = \perp$ then return $\perp$
  $\boldsymbol{A}'_j \leftarrow \boldsymbol{A}'_j \| A$; $\boldsymbol{C}'_j \leftarrow \boldsymbol{C}'_j \| C$
  if $H[N'_j, \boldsymbol{A}'_j, \boldsymbol{C}'_j, 0] \neq \perp$ then
    $\boldsymbol{M}'_j \leftarrow \boldsymbol{M} \leftarrow H[N'_j, \boldsymbol{A}'_j, \boldsymbol{C}'_j, 0]$
    return $\boldsymbol{M}[|\boldsymbol{M}|]$
  $M \leftarrow \rho^{-1}_{N'_j, \boldsymbol{A}'_j, \boldsymbol{M}'_j, 0}(C)$
  if $M = \perp$ then $S'_j \leftarrow \perp$
  else $\boldsymbol{M}'_j \leftarrow \boldsymbol{M}'_j \| M$ fi
  return $M$

proc Dec.last($j, A, C$)
  if $j > J$ or $S'_j = \perp$ then return $\perp$
  $\boldsymbol{A}'_j \leftarrow \boldsymbol{A}'_j \| A$; $\boldsymbol{C}'_j \leftarrow \boldsymbol{C}$; $S'_j \leftarrow \perp$
  if $H[N'_j, \boldsymbol{A}'_j, \boldsymbol{C}'_j, 0] \neq \perp$ then
    $\boldsymbol{M} \leftarrow H[N'_j, \boldsymbol{A}'_j, \boldsymbol{C}'_j, 1]$
    return $\boldsymbol{M}[|\boldsymbol{M}|]$
  $M \leftarrow \rho^{-1}_{N'_j, \boldsymbol{A}'_j, \boldsymbol{M}'_j, 1}(C)$
  return $M$
```

Figure 7.10 – **Games G_1–G_3 in the proof of Proposition 7.6.** Game G_2 contains the boxed statements, but game G_1 doesn't. There is an implicit procedure **initialize**() that initializes $I, J \leftarrow 0$ and $Z \leftarrow \emptyset$, and samples $\rho_{N,\boldsymbol{A},\boldsymbol{M},\delta} \leftarrow\!\!\$\ \text{Inj}(\tau)$ for every $N \in \mathcal{N}, \delta \in \{0,1\}$' and $\boldsymbol{A}, \boldsymbol{M} \in \{0,1\}^{**}$ such that $|\boldsymbol{A}| = |\boldsymbol{M}| + 1$.

have only p elements that correspond to the p decryption chains. Adversary $\mathscr{B}_2$ then outputs a random element of $\mathcal{Y}$ as its forgery attempt.

Consider games G_1–G_3 in Figure 7.10. Game G_1 corresponds to game $\mathbf{oae2b\text{-}R}_\Pi$. Game G_2 is identical to game G_1, except that Dec.next and Dec.last always return $\perp$. The two games are identical-until-bad, and thus by Lemma 2.1

$$\begin{aligned}\Pr[\mathscr{A}^{G_1} \Rightarrow 1] - \Pr[A^{G_2} \Rightarrow 1] &\leq \Pr[\mathscr{A}^{G_2} \text{ sets } \mathsf{bad}] \\ &\leq p \cdot \Pr[\mathscr{B}_2^{\mathbf{oae2c\text{-}F}_\Pi}] \\ &\leq p \cdot \mathbf{Adv}_\Pi^{\mathbf{oae2\text{-}auth}}(\mathscr{B}_2) \ .\end{aligned}$$

Game G_3 is identical to game G_2, except that instead of calling Enc.next$(i, A, \cdot)$ and Enc.last$(i, A, \cdot)$, we use $\rho_{N_i,\boldsymbol{A}_i,\boldsymbol{M}_i,0} \leftarrow\$ \operatorname{Inj}(\tau)$ and $\rho_{N_i,\boldsymbol{A}_i,\boldsymbol{M}_i,1} \leftarrow\$ \operatorname{Inj}(\tau)$ respectively. Moreover, Dec.next$(i, A, \cdot)$ and Dec.last$(i, A, \cdot)$ are also replaced by $\rho^{-1}_{N_i,\boldsymbol{A}_i,\boldsymbol{M}_i,0}$ and $\rho^{-1}_{N_i,\boldsymbol{A}_i,\boldsymbol{M}_i,1}$ respectively.

We have $\Pr[\mathscr{A}^{G_2} \Rightarrow 1] = \Pr[\mathscr{B}_1^{\mathbf{oae2c\text{-}R}} \Rightarrow 1]$. In addition, $\Pr[\mathscr{B}_1^{\mathbf{oae2c\text{-}I}} \Rightarrow 1] - \Pr[\mathscr{A}^{G_3} \Rightarrow 1]$ is exactly the gap between PRI and MRAE, which is upper-bounded by $q^2/2^{\tau+1} + 4q/2^\tau \leq q^2/2^\tau$ [RS06b, Theorem 7]. Finally, game G_3 coincides with game $\mathbf{oae2b\text{-}I}_\Pi$. Summing up,

$$\begin{aligned}\mathbf{Adv}_\Pi^{\mathbf{oae2b}}(\mathscr{A}) &= \Pr[A^{G_1} \Rightarrow 1] - \Pr[A^{G_3} \Rightarrow 1] \\ &\leq \mathbf{Adv}_\Pi^{\mathbf{oae2\text{-}priv}}(\mathscr{B}_1) + p \cdot \mathbf{Adv}_\Pi^{\mathbf{oae2\text{-}auth}}(\mathscr{B}_2) + q^2/2^\tau \ .\end{aligned}$$

□

Proposition 7.7 (OAE2b $\Rightarrow$ OAE2c)**.** *Let Π be a segmented-AE scheme with ciphertext expansion τ. For any adversaries $\mathscr{A}_1$ that runs in time $t+1$ and makes no more than q individual queries with σ_1 bits of data in total, and $\mathscr{A}_2$ that runs in time t_2, queries at most σ_2 bits of data in total and outputs ℓ segments in its forgery attempt, there exist adversaries $\mathscr{B}_1$ and $\mathscr{B}_2$ for which*

$$\begin{aligned}&\mathbf{Adv}_\Pi^{\mathbf{oae2\text{-}priv}}(\mathscr{A}_1) \leq \mathbf{Adv}_\Pi^{\mathbf{oae2b}}(\mathscr{B}_1) + q^2/2^\tau \ \textit{and} \\ &\mathbf{Adv}_\Pi^{\mathbf{oae2\text{-}auth}}(\mathscr{A}_2) \leq \mathbf{Adv}_\Pi^{\mathbf{oae2b}}(\mathscr{B}_2) + \ell/2^\tau.\end{aligned}$$

For each $i \in \{1, 2\}$, adversary $\mathscr{B}_i$ runs in time $t_i + \gamma_i \cdot \sigma_i$ for some constant γ_i, and the total length of its queries is at most σ_i.

Proof. Constructing $\mathscr{B}_1$ is trivial: it ignores its decryption oracles and runs $\mathscr{A}_1$ on its encryption oracles. Then $\Pr[\mathscr{B}_1^{\mathbf{oae2b\text{-}R}_\Pi} \Rightarrow 1] = \Pr[\mathscr{A}_1^{\mathbf{oae2c\text{-}R}_\Pi} \Rightarrow 1]$, and $\Pr[\mathscr{B}_1^{\mathbf{oae2b\text{-}I}} \Rightarrow 1] - \Pr[\mathscr{A}_1^{\mathbf{oae2c\text{-}I}} \Rightarrow 1]$ is the gap between PRI and MRAE, which is upper-bounded by $q^2/2^{\tau+1} + 4q/2^\tau \leq q^2/2^\tau$ [RS06b, Theorem 7]. Hence

$$\mathbf{Adv}_\Pi^{\mathbf{oae2b}}(\mathscr{B}_1) = \mathbf{Adv}_\Pi^{\mathbf{oae2\text{-}priv}}(\mathscr{A}_1) + q^2/2^\tau.$$

We create the adversary $\mathscr{B}_2$ from $\mathscr{A}_2$ as follows. The former runs the latter on its encryption oracles and maintains the set $\mathcal{Y}$ of the partial decryption chains $(N_i, \boldsymbol{A}_i, \boldsymbol{C}_i, \delta_i)$ as in game **oae2c-F**$_\Pi$. When $\mathscr{A}_2$ outputs $(N, \boldsymbol{A}, \boldsymbol{C}, b)$, adversary $\mathscr{B}_2$ runs the following code:

```
if $|\boldsymbol{A}| \neq |\boldsymbol{C}|$ or $(N, \boldsymbol{A}, \boldsymbol{C}, b) \in \mathcal{Y}$ or $|\boldsymbol{C}| = 0$ then return 0
Dec.init$(N)$; $m \leftarrow |\boldsymbol{C}|$
for $i \leftarrow 1$ to $m - b$ do
    $(M, S) \leftarrow$ Dec.next$(1, \boldsymbol{A}[i], \boldsymbol{C}[i])$
    if $M = \perp$ then return 0
if $b = 1$ and Dec.last$(1, \boldsymbol{A}[m], \boldsymbol{C}[m]) = \perp$ then return 0
return 1
```

Then $\Pr[\mathscr{B}_2^{\textbf{oae2b-I}_\Pi} \Rightarrow 1] \leq \ell/2^\tau$ and $\mathbf{Adv}_\Pi^{\textbf{oae2-auth}}(\mathscr{A}_2) = \Pr[\mathscr{B}_2^{\textbf{oae2b-R}_\Pi} \Rightarrow 1]$. Hence $\mathbf{Adv}_\Pi^{\textbf{oae2-auth}}(\mathscr{A}_2) \leq \mathbf{Adv}_\Pi^{\textbf{oae2b}}(\mathscr{B}_2) + \ell/2^\tau$. □

7.5.4 Discussion

Multivalued segment-expansion. It is easy to extend the definitions of this section to schemes for which the segment-expansion varies according to segment position. In particular, one could use one expansion value, σ, for plaintext components other than the last, and a different expansion value, τ, at the end. For such a (σ, τ)-expanding scheme, distribution IdealOAE(τ) would be adjusted to IdealOAE(σ, τ) in the natural way.

The main reason for considering multivalued segment-expansion is to clarify how OAE2 security relates to prior notions in the literature. In particular, OAE2 resembles OAE1 where the segment-expansion is $(0, \tau)$ and where all segments are required to have some fixed length n. Yet even then the definitions would be very different: the OAE2 version would be stronger, since an online decryption capability is not allowed to compromise OAE2 security, whereas the capability may compromise OAE1 security. It is easy to give a separating example; see Appendix B.3.

Another potential reason to consider multivalued segment-expansion is as a way to save on bits; obviously one will use fewer total bits, over a sequence of two or more segments, if only the last is expanded. But we suspect that this benefit is rarely worth its cost. If segments are 1 KByte (which is fairly short) and tags are 128 bits (which is fairly long), the difference (in total number of needed bits) between authenticating every segment and authenticating only the last one will always be less than 2%. In most application, this seems a small price to pay to have each and every segment properly authenticated.[7]

Why vector-valued AD. When modelling OAE, we were unsure if one ought to think of the AD as a fixed string that is known before the plaintext begins to arrive, or if, instead, one should think of the AD as vector-valued, its i^{th} segment available when the i^{th} segment of plaintext is. We adopted the second view (switching from the first at

[7]At the same time, we recognize that in some *very* constrained applications, e.g. battery-powered sensors, saving 2% on communication complexity may lead to a non-negligible increase of battery life.

the urging of the Keyak team) for closer concordance with prior work [BDPA11a] and for greater generality: a string-valued AD of A can be regarded as a vector-valued AD of $\boldsymbol{A} = (A, \varepsilon, \varepsilon, \ldots)$. More philosophically, the two conceptions correspond to whether one thinks of breaking up a fixed plaintext $\boldsymbol{M}$ into a sequence of segments M_i or one regards the M_i values as more autonomous, each encrypted when available, each with its own associated context. With plaintexts and AD both vector-valued, one conceptually extends across time a channel that securely transmit pairs of strings, one component with privacy and both with authenticity. All that said, the actual utility of the vector-valued choice over string-valued AD is uncertain.

The impact of nonce reuse. One of our main objections against FFL's notion of OAE1 was that it was advertised as capturing nonce-misuse resistance, while the nonce-reusing CPSS attack suggests that this label is not that accurate. But the CPSS attack, once accordingly adjusted, also applies to any OAE2-secure scheme. It applies even if the attacker does not control the segmentation, it is enough that a plaintext always gets segmented with the same pattern, starting from the left.

In fact, if the encryption is online, and if the setting is such that a secret value is prepended with a prefix that is under adversarial control, and if nonce-reuse can occur, a variant of the CPSS attack will apply. It easy to see why; using the controlled prefix, the adversary can always shift the secret around the boundaries of ciphertext-segments to apply a divide-and-conquer brute force strategy, as long as the segmentation pattern is sufficiently predictable and repetitive.

This is why we refrain from referring to OAE2 as a notion for nonce-misuse resistance: because no AE scheme with online encryption can deliver on the intuition of resisting the nonce reuse. We prefer the term "best-possible security."

7.6 Achieving OAE2

In the special case that each segmented-string has only one component, OAE2 (specifically OAE2a and OAE2b) degenerates to the notion of a τ-expanding pseudorandom injection (Definition 2.11). The notion is close to MRAE (Definition 2.10), with a gap $q^2/2^{s+\tau} + q/2^\tau$ where q is the number of queries and s is the length of the shortest plaintext queried. We construct an OAE2-secure scheme *from* a PRI-secure scheme. The scheme could be SIV [RS06b] if τ is large, say $\tau = 128$, or AEZ scheme [HKR15], for arbitrary τ.

The construction. Fix integers $n \geq \tau \geq 0$. Let $\langle \cdot \rangle$ denote an efficient injective encoding that maps a pair $(A, d) \in \{0,1\}^* \times \{0, 1, 2, 3, 4\}$ to a string $\langle A, d \rangle \in \{0,1\}^*$. For example, one can represent d by a three-bit string, and append this to A. Let $\Pi = (\mathbf{K}, \mathbf{E}, \mathbf{D})$ be a nonce-based AE scheme of ciphertext-expansion τ, nonce space $\{0,1\}^n$, and AD space $\{0,1\}^*$. Figure 7.11 defines a segmented-AE scheme $\mathbf{CHAIN}[\Pi, \langle \cdot \rangle, n] =$

$(\mathcal{K}, \mathcal{E}, \mathcal{D})$ with segment expansion τ, nonce space $\{0,1\}^n$, AD space $\{0,1\}^*$, and state space $\mathbf{K} \times \{0,1\}^n$.

The intuition behind the construction is best characterized by its name. A sequence of $\mathcal{E}$.next calls terminated by an $\mathcal{E}$.last call is implemented as a sequence of evaluations of the nonce-based AE-encryption $\mathbf{E}$, *chained* by a value derived from each plaintext and ciphertext segment that is fed to the nonce input. We use the AD-input of $\mathbf{E}$ to ensure proper domain separation between the $\mathcal{E}$.next and $\mathcal{E}$.last calls. We give a formal statement about the OAE2 security of CHAIN in Theorem 7.8.

Remark 7 (A flaw in **CHAIN**.). *During the writing of this dissertation, a flaw in the* **CHAIN** *construction (call the flawed version* **CHAIN′***) was discovered by its authors. More precisely, the domain separation constants "d" that get encoded into the AD input of the underlying PRI did not differentiate the processing of the first segment in a chain from the processing of all the other "interior" segments. This gave rise to an attack that easily distinguishes* **CHAIN′** *from the idealized reference objects with two queries.*

The key property used in the attack is that whenever the segmented plaintext $\boldsymbol{M} = (0^n, 0^n, 0^n)$ *got encrypted to* $\boldsymbol{C} = (\boldsymbol{C}[1], \boldsymbol{C}[2], \boldsymbol{C}[3])$ *(with some nonce and empty AD segments) by* **CHAIN′***, then the segmented plaintext* $\boldsymbol{M} = (0^n, 0^n)$ *encrypted with the nonce* $\boldsymbol{C}[1]$ *(and empty AD segments) necessarily got encrypted to* $\boldsymbol{C} = (\boldsymbol{C}[2], \boldsymbol{C}[3])$.

In this chapter, we present the corrected construction.

Discussion. In $\mathcal{E}$.next and $\mathcal{D}$.next, the state is computed via $\mathsf{left}_n(M) \oplus \mathsf{left}_n(C)$. One might instead xor the n-bit suffix of M and C; this makes no difference. On the other hand, suppose one uses *just* $\mathsf{left}_n(C)$, eliminating the xor with $\mathsf{left}_n(M)$. Call this variant $\mathbf{CHAIN1}[\Pi, \langle\cdot,\rangle, n]$. The method is insecure for small τ. Here is an attack for the case $\tau = 0$. The adversary makes a single query $(N, \boldsymbol{A}, \boldsymbol{C})$ to the decryption oracle, where N is arbitrary, $\boldsymbol{A} = (\varepsilon, \varepsilon, \varepsilon)$ and $\boldsymbol{C} = (0^n, 0^n, 0^n, 0^n)$. Let the answer be $\boldsymbol{M} = (M_1, M_2, M_3, M_4)$. The adversary will output 1 only if $M_2 = M_3$. In the **oae2b-I** game the strings M_2 and M_3 are independent random strings. However, in game **oae2b-R** we always have $M_2 = M_3 = \mathbf{D}_K(0^n, \langle\varepsilon, 0\rangle, 0^n)$. Hence the adversary can win with advantage $1 - 2^{-n}$. In contrast, for large τ, scheme $\mathbf{CHAIN1}[\Pi, \langle\cdot,\rangle, n]$ *is* OAE2 secure.

To achieve OAE2 with multivalued segment-expansion, use an RAE-secure underlying scheme [HKR15], a generalization of PRI that allows one to select an arbitrary ciphertext-expansion for each query. The construction is modified in the natural way.

Theorem 7.8. *Let* Π *be a nonce-based AE scheme with stretch* τ. *Let further* $\langle\cdot\rangle : \{0,1\}^* \times \{0,1,2,3,4\} \to \{0,1\}^*$ *be an efficient injective encoding, let* $n \geq \tau$, *and let* $\mathbf{CHAIN}[\Pi, \langle\cdot\rangle, n]$ *be as defined in Figure 7.11. Let* $\mathscr{A}$ *be an adversary that runs in time* t, *makes no more than* $q \leq 2^{n-1}$ *queries in total, such that their data complexity is limited by* σ *bits. Then*

$$\mathbf{Adv}^{\mathbf{oae2b}}_{\mathbf{CHAIN}[\Pi,\langle\cdot\rangle,n]}(\mathscr{A}) \leq \mathbf{Adv}^{\mathbf{pri}}_{\Pi}(\mathscr{B}) + 2q^2/2^n$$

```
1: algorithm $\mathcal{E}$.init$(K, N)$
2:     return $(K, N, 0)$
3: end algorithm

1: algorithm $\mathcal{E}$.next$(S, A, M)$
2:     $(K, V, d) \leftarrow S$
3:     $C \leftarrow \mathbf{E}_K(V, \langle A, d\rangle, M)$
4:     if $|M| \geq n$ then
5:         $V \leftarrow (\mathsf{left}_n\,(C) \oplus \mathsf{left}_n\,(M))$
6:     else
7:         $V \leftarrow \mathsf{left}_n\,(\mathbf{E}_K(V, \langle A, 3\rangle, M \| 0^n))$
8:     end if
9:     return $(C, (K, V, 1))$
10: end algorithm

1: algorithm $\mathcal{E}$.last$(S, A, M)$
2:     $(K, V, d) \leftarrow S$
3:     if $d = 0$ then
4:         $d \leftarrow 4$
5:     else
6:         $d \leftarrow 2$
7:     end if
8:     return $\mathbf{E}_K(V, \langle A, d\rangle, M)$
9: end algorithm
```

```
1: algorithm $\mathcal{D}$.init$(K, N)$
2:     return $(K, N, 0)$
3: end algorithm

1: algorithm $\mathcal{D}$.next$(S, A, C)$
2:     $(K, V, d) \leftarrow S$
3:     $M \leftarrow \mathbf{D}_K(V, \langle A, d\rangle, C)$
4:     if $M = \perp$ then
5:         return $(\perp, \perp)$
6:     end if
7:     if $|M| \geq n$ then
8:         $V \leftarrow \mathsf{left}_n\,(C) \oplus \mathsf{left}_n\,(M)$
9:     else
10:        $V \leftarrow \mathsf{left}_n\,(\mathbf{E}_K(V, \langle A, 3\rangle, M \| 0^n))$
11:    end if
12:    return $(M, (K, V, 1))$
13: end algorithm

1: algorithm $\mathcal{D}$.last$(S, A, C)$
2:     $(K, V, d) \leftarrow S$
3:     if $d = 0$ then
4:         $d \leftarrow 4$
5:     else
6:         $d \leftarrow 2$
7:     end if
8:     return $\mathbf{D}_K(V, \langle A, d\rangle, C)$
9: end algorithm
```

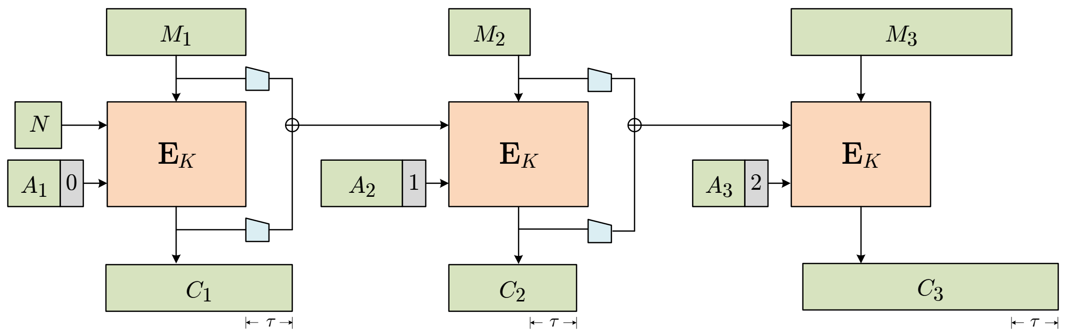


Figure 7.11 – **The CHAIN construction for OAE2. Top:** Encryption scheme $\Pi = (\mathbf{K}, \mathbf{E}, \mathbf{D})$, secure as a PRI with expansion τ, is turned into a segmented-AE scheme $\mathbf{CHAIN}[\Pi, \langle\cdot\rangle, n] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with $\mathcal{K} = \mathbf{K}$. **Bottom:** Illustration of the scheme. Each segment of (M_1, M_2, M_3) has at least n bits. Trapezoids represent truncation to n bits.

for some $\mathscr{B}$ that runs in time $t + \gamma \cdot qn$ for some constant γ, and makes no more than $2q$ queries that have a total data complexity limited by $5qn$ bits.

Proof. We construct the adversary $\mathscr{B}$ from $\mathscr{A}$ as follows. The former runs the latter and simulates game $\mathbf{oae2b\text{-}R}_{\mathbf{CHAIN}[\Pi,\langle\cdot\rangle,n]}$, but each call to $\mathbf{E}_K(\cdot)$ or $\mathbf{D}_K(\cdot)$ is replaced by the corresponding query to the Enc or Dec oracle of $\mathscr{B}$, respectively. Adversary $\mathscr{B}$ then outputs the same guess as $\mathscr{A}$.

Consider the games G_1–G_3 in Figure 7.12. Game G_1 is identical with the game $\mathbf{oae2b\text{-}R}_{\mathbf{CHAIN}[\Pi,\langle\cdot\rangle,n]}$. We now analyse the transitions between the games. In all three games, we sample $\pi_{V,\langle A',3\rangle} \leftarrow\$ \operatorname{Inj}(\tau)$ for every $V \in \{0,1\}^n$ and every $A' \in \{0,1\}^*$, and $\rho_{N,\boldsymbol{A},\boldsymbol{M},\delta} \leftarrow\$ \operatorname{Inj}(\tau)$ for every $N \in \{0,1\}^n, \delta \in \{0,1\}$, and $\boldsymbol{A}, \boldsymbol{M} \in \{0,1\}^{**}$ such that $|\boldsymbol{A}| = |\boldsymbol{M}| + 1$ in the procedure **initialize**.

Game G_2 is identical to game G_1, except that instead of using $\mathbf{E}_K^{V,A}$ and $\mathbf{D}_K^{V,A}$, we'll use an injective function $\pi_{V,A}$ and its inverse $\pi_{V,A}^{-1}$ respectively. We ensure that $\pi_{V,A} \leftarrow\$ \operatorname{Inj}(\tau)$: If $\pi_{V,A}$ is not defined, we'll implement it via $\rho_{N,\boldsymbol{A},\boldsymbol{M},\delta}$, where $(N, \boldsymbol{A}, \boldsymbol{M}, \delta)$ is created right before we call $\operatorname{Map}(V, A, M)$ or $\operatorname{MapInv}(V, A, C)$. Note that $\pi_{V,A}$ will be independent for each (V, A), because different pairs (V, A), because (V, A) is computed as a function of $(N, \boldsymbol{A}, \boldsymbol{M}, \delta)$ in G_2, so $(V, A) \neq (V', A')$ will necessarily have two distinct preimages $(N, \boldsymbol{A}, \boldsymbol{M}, \delta) \neq (N', \boldsymbol{A}', \boldsymbol{M}', \delta')$. Then if $\mathscr{B}$ plays **pri-I** it perfectly simulates G_2 for $\mathscr{A}$ and we have

$$\Pr[\mathscr{A}^{G_1} \Rightarrow 1] - \Pr[\mathscr{A}^{G_2} \Rightarrow 1] = \mathbf{Adv}_{\Pi}^{\mathbf{pri}}(\mathscr{B}) \ .$$

Game G_3 is identical to game G_2 except that, we make sure that the state V' never repeats, while maintaining the following consistency: (i) calling Map with the same (V, A, M) always results in the same (C, V'), (ii) calling MapInv with the same (V, A, C) always result in the same (M, V'), and (iii) if $(C, V') \leftarrow \operatorname{Map}(V, A, M)$ then necessarily $\operatorname{MapInv}(V, A, C)$ returns (M, V'), and (iv) if $(M, V') \leftarrow \operatorname{MapInv}(V, A, C)$ and $M \neq \perp$ then $\operatorname{Map}(V, A, M)$ returns (C, V'). The two games are identical-until-bad, and thus (by Lemma 2.1)

$$\Pr[\mathscr{A}^{G_2} \Rightarrow 1] - \Pr[\mathscr{A}^{G_3} \Rightarrow 1] \leq \Pr[G_2 \text{ sets } \mathsf{bad}] \ .$$

We now bound the chance that game G_2 sets bad. Terminate the game immediately when bad gets set; it doesn't change the probability that G_2 sets bad. Observe that

(1) In $\operatorname{Map}(V, A, M)$ and $\operatorname{MapInv}(V, A, C)$, we'll have A of the form $\langle A', d\rangle$, with $d \in \{0, 1, 2, 4\}$.

(2) In $\operatorname{Map}(V, A, M)$, we compute $C \leftarrow \pi_{V,A}(M)$ and invoke Ev to compute $L \leftarrow \pi_{V,\langle A',3\rangle}(M \| 0^n)$. The second call is made only if $|M| < n$, $d \in \{0, 1, 2, 4\}$, and there is no prior call $\operatorname{Map}(V, A, M)$ or $\operatorname{MapInv}(V, A, C)$.

(3) In $\operatorname{MapInv}(V, A, C)$, we compute $M \leftarrow \pi_{V,A}^{-1}(C)$ and invoke Ev to compute $L \leftarrow \pi_{V,\langle A',3\rangle}(M \| 0^n)$. The second call is made only if $|M| < n$, $d \in \{0, 1, 2, 4\}$, and

```
proc Enc.init($N$)
$I \leftarrow I+1$; $\boldsymbol{M}_I \leftarrow \Lambda$; $\boldsymbol{A}_I \leftarrow \Lambda$
$S_I \leftarrow (K, N, 0)$; $N_i \leftarrow N$
return $I$

proc Enc.next($i, A, M$)
if $i > I$ or $S_i = \bot$ then return $\bot$
$(K, V, d) \leftarrow S_i$; $\boldsymbol{A}_i \leftarrow \boldsymbol{A}_i \| A$
$(N, \boldsymbol{A}, \boldsymbol{M}, \delta) \leftarrow (N_i, \boldsymbol{A}_i, \boldsymbol{M}_i, 0)$
$(C, V) \leftarrow \mathrm{Map}(V, \langle A, d\rangle, M)$
$S_i \leftarrow (K, V, 1)$; $\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i \| M$
return $C$

proc Enc.last($i, A, M$)
if $i > I$ or $S_i = \bot$ then return $\bot$
$(K, V, d) \leftarrow S_i$; $\boldsymbol{A}_i \leftarrow \boldsymbol{A}_i \| A$
if $d = 0$ then $d \leftarrow 4$ else $d \leftarrow 2$
$(N, \boldsymbol{A}, \boldsymbol{M}, \delta) \leftarrow (N_i, \boldsymbol{A}_i, \boldsymbol{M}_i, 1)$
$(C, V) \leftarrow \mathrm{Map}(V, \langle A, d\rangle, M)$; $S_i \leftarrow \bot$
return $C$
```

```
proc Dec.init($N$)
$J \leftarrow J+1$; $\boldsymbol{M}'_J \leftarrow \Lambda$; $\boldsymbol{A}'_J \leftarrow \Lambda$
$S'_J \leftarrow (K, N, 0)$; $N'_j \leftarrow N$
return $J$

proc Dec.next($j, A, C$)
if $j > J$ or $S'_j = \bot$ then return $\bot$
$(K, V, d) \leftarrow S'_j$; $\boldsymbol{A}'_j \leftarrow \boldsymbol{A}'_j \| A$
$(N, \boldsymbol{A}, \boldsymbol{M}, \delta) \leftarrow (N'_j, \boldsymbol{A}'_j, \boldsymbol{M}'_j, 0)$
$(M, V) \leftarrow \mathrm{MapInv}(V, \langle A, d\rangle, C)$
if $V \neq \bot$ then
    $S'_j \leftarrow (K, V, 1)$
else
    $S'_j \leftarrow \bot$
$\boldsymbol{M}'_j \leftarrow \boldsymbol{M}'_j \| M$
return $M$

proc Dec.last($j, A, C$)
if $j > J$ or $S_j = \bot$ then return $\bot$
$(K, V, d) \leftarrow S'_j$; $\boldsymbol{A}_j \leftarrow \boldsymbol{A}_j \| A$
$(N, \boldsymbol{A}, \boldsymbol{M}, \delta) \leftarrow (N'_j, \boldsymbol{A}'_j, \boldsymbol{M}'_j, 1)$
if $d = 0$ then $d \leftarrow 4$ else $d \leftarrow 2$
$(M, V) \leftarrow \mathrm{MapInv}(V, \langle A, d\rangle, C)$
$S'_j \leftarrow \bot$
return $M$
```

```
proc Map($V, A, M$)                                   Game $G_1$
$C \leftarrow \mathbf{E}_K^{V,A}(M)$
if $H[V, A, M] \neq \bot$ then $V' \leftarrow H[V, A, M]$
else $V' \leftarrow \mathrm{Ev}(V, A, M, C)$; $H[V, A, M] \leftarrow V'$
return $(C, V')$

proc MapInv($V, A, C$)
$M \leftarrow \mathbf{D}_K^{V,A}(C)$
if $M = \bot$ then return $(\bot, \bot)$
if $H[V, A, M] \neq \bot$ then $V' \leftarrow H[V, A, M]$
else $V' \leftarrow \mathrm{Ev}(V, A, M, C)$; $H[V, A, M] \leftarrow V'$
return $(M, V')$

proc Ev($V, A, M, C$)
if $|M| \geq n$ then $V' \leftarrow \mathsf{left}_n(C) \oplus \mathsf{left}_n(M)$
elsif $A = \langle A', d\rangle$ then
    $L \leftarrow \mathbf{E}_K(V, \langle A', 3\rangle, M \| 0^n)$; $V' \leftarrow \mathsf{left}_n(L)$
Dom $\leftarrow$ Dom $\cup \{V'\}$
return $V'$
```

```
proc Map($V, A, M$)                         Games $G_2$, $\boxed{G_3}$
if $\pi_{V,A} = \bot$ then $\pi_{V,A} \leftarrow \rho_{N,\boldsymbol{A},\boldsymbol{M},\delta}$
$C \leftarrow \pi_{V,A}(M)$
if $H[V, A, M] \neq \bot$ then $V' \leftarrow H[V, A, M]$
else $V' \leftarrow \mathrm{Ev}(V, A, M, C)$; $H[V, A, M] \leftarrow V'$
return $(C, V')$

proc MapInv($V, A, C$)
if $\pi_{V,A} = \bot$ then $\pi_{V,A} \leftarrow \rho_{N,\boldsymbol{A},\boldsymbol{M},\delta}$
$M \leftarrow \pi_{V,A}^{-1}(C)$
if $M = \bot$ then return $(\bot, \bot)$
if $H[V, A, M] \neq \bot$ then $V' \leftarrow H[V, A, M]$
else $V' \leftarrow \mathrm{Ev}(V, A, M, C)$; $H[V, A, M] \leftarrow V'$
return $(M, V')$

proc Ev($V, A, M, C$)
if $|M| \geq n$ then $V' \leftarrow \mathsf{left}_n(C) \oplus \mathsf{left}_n(M)$
elsif $A = \langle A', d\rangle$ then
    $L \leftarrow \pi_{V,\langle A', 3\rangle}(M \| 0^n)$; $V' \leftarrow \mathsf{left}_n(L)$
if ($V' \in$ Dom) then
    bad $\leftarrow$ true; $\boxed{V' \leftarrow\!\!\$\ \{0,1\}^n \backslash \mathrm{Dom}}$
Dom $\leftarrow$ Dom $\cup \{V'\}$
return $V'$
```

Figure 7.12 – **Games G_1–G_3 used in the proof of Theorem 7.8.** Game G_3 contains the corresponding boxed statements but game G_2 doesn't. The games share the common procedures Enc.init, Enc.next, Enc.last, Dec.init, Dec.next, and Dec.last, and each game uses private procedures Map, MapInv, and Ev that are inaccessible to the adversary. In each game, there is an implicit procedure **initialize**() that initializes Dom $\leftarrow \emptyset$ and $I, J \leftarrow 0$, and samples $K \leftarrow\!\!\$\ \mathbf{K}$, $\pi_{V,\langle A',3\rangle} \leftarrow\!\!\$\ \mathrm{Inj}(\tau)$ for every $V \in \{0,1\}^n$ and every $A' \in \{0,1\}^*$, and $\rho_{N,\boldsymbol{A},\boldsymbol{M},\delta} \leftarrow\!\!\$\ \mathrm{Inj}(\tau)$ for every $N \in \{0,1\}^n, \delta \in \{0,1\}$, and $\boldsymbol{A}, \boldsymbol{M} \in \{0,1\}^{**}$ such that $|\boldsymbol{A}| = |\boldsymbol{M}| + 1$.

there is no prior call $\mathrm{Map}(V, A, M)$ or $\mathrm{MapInv}(V, A, C)$.

Recall that $1 < q \le 2^{n-1}$. otherwise the bound is trivial. The flag bad is triggered only if the state V' repeats one of its prior values in the Ev procedure In the following case analysis we bound the probability that the i^{th} value V' computed in G_2 triggers bad, i.e. that the value V' sampled in i^{th} query falls into Dom.

Case 1: $V' \leftarrow \mathsf{left}_n(L)$, where $L \leftarrow \pi_{V,\langle A',3\rangle}(M\|0^n)$. From (1), (2), and (3), since there is a one-to-one correspondence between M and $M\|0^n$, there is no prior call to $\pi_{V,\langle A',3\rangle}(M\|0^n)$ or $\pi^{-1}_{V,\langle A',3\rangle}(L)$. Then L is chosen uniformly random from a subset of $\{0,1\}^{n+\tau}$ that has at least $2^{n+\tau} - q$ elements and $|\mathrm{Dom}| \le (i-1)$. The collision of the i^{th} V' in this case occurs with probability bounded by $(i-1)/(2^n - q)$.

Case 2: $V' \leftarrow \mathsf{left}_n(C) \oplus \mathsf{left}_n(M)$. There must be no prior Map call of the same (V, A, M) or MapInv call of the same (V, A, C), otherwise we'll return the consistent state, and bad won't be triggered. First suppose that V' collides during the execution of Map. Let $s = |C|$. From (1), (2), and (3), there is no prior call to $\pi_{V,A}(M)$ or $\pi^{-1}_{V,A}(C)$. We have $|\mathrm{Dom}| \le (i-1)$. Since we sample C uniformly from a subset of $\{0,1\}^s$ that has at least $2^s - q$ elements, the collision of i^{th} V' occurs with probability at most $(i-1)2^{s-n}/(2^s - q) \le 2(i-1)/(2^n - q)$. Next, consider the case that V' collides during the execution of MapInv. Again, we have $|\mathrm{Dom}| \le (i-1)$. Let $s = |M|$. By using the same analysis as above, the probability of $V' \in \mathrm{Dom}$ is bounded by $(i-1)/(2^n - q)$.

Because there will be no more than q values V' sampled during the game, by union bound the probability that bad is triggered is bounded by

$$\sum_{i=1}^{q} \frac{2(i-1)}{2^n - q} \le \frac{q^2}{2^n - q} \le \frac{2q^2}{2^n};$$

the last inequality is due to the assumption that $q \le 2^{n-1}$.

In game G_3, distinct tuples $(N, \boldsymbol{A}, \boldsymbol{M}, \delta)$ will correspond to different pairs (V, A); we will justify the claim later. Hence $\pi_{V,A}$ in Map/MapInv coincides with $\rho_{N,\boldsymbol{A},\boldsymbol{M},\delta}$. Then game G_3 is equivalent to game $\mathbf{oae2b\text{-}I}_{\mathbf{CHAIN}[\Pi,\langle\cdot\rangle,n]}$, and thus

$$\mathbf{Adv}^{\mathbf{oae2b}}_{\mathbf{CHAIN}[\Pi,\langle\cdot\rangle,n]}(\mathscr{A}) = \Pr[\mathscr{A}^{G_1} \Rightarrow 1] - \Pr[\mathscr{A}^{G_3} \Rightarrow 1] \le \mathbf{Adv}^{\mathbf{pri}}_{\Pi}(\mathscr{B}) + \frac{2q^2}{2^n}.$$

What remains is to justify the claim above. Suppose that there exists at least one pair of distinct tuples $(N_1, \boldsymbol{A}_1, \boldsymbol{M}_1, \delta_1)$ and $(N_2, \boldsymbol{A}_2, \boldsymbol{M}_2, \delta_2)$ that correspond to the same pair (V, A). Among such pairs of tuples, consider the one that minimizes $|\boldsymbol{M}_1|$. Consider the following cases.

Case 1: $|\boldsymbol{M}_1|, |\boldsymbol{M}_2| > 0$. Let $\boldsymbol{M}_1^*$ be the prefix of $\boldsymbol{M}^*$ that consists of $|\boldsymbol{M}_1| - 1$ components. Define $\boldsymbol{M}_2^*$ for $\boldsymbol{M}_2$, $\boldsymbol{A}_1^*$ for $\boldsymbol{A}_1$, and $\boldsymbol{A}_2^*$ for $\boldsymbol{A}_2^*$ analogously. Then it means that $(N_1, \boldsymbol{A}_1^*, \boldsymbol{M}_1^*, 0)$ and $(N_2, \boldsymbol{A}_2^*, \boldsymbol{M}_1^*, 0)$ correspond to the same pair (V^*, A^*) as well, but that contradicts the minimum of $|\boldsymbol{M}_1|$.

Case 2: $|\boldsymbol{M}_1| = 0$ and $|\boldsymbol{M}_2| > 0$. But then this is a contradiction: (1) since $(N_1, \boldsymbol{A}_1, \boldsymbol{M}_1, \delta_1)$ corresponds to (V, A), it means that A is of the form $\langle A', 0\rangle$ or $\langle A', 4\rangle$, but (2) since $(N_2, \boldsymbol{A}_2, \boldsymbol{M}_2, \delta_2)$ corresponds to (V, A), it means that A is of the form $\langle A'', 1\rangle$ or $\langle A'', 2\rangle$.

Case 3: $|\boldsymbol{M}_1| > 0$ and $|\boldsymbol{M}_2| = 0$. This is similar to Case 2.

Case 4: $|\boldsymbol{M}_1| = |\boldsymbol{M}_2| = 0$, meaning that $\boldsymbol{M}_1 = \boldsymbol{M}_2 = \Lambda$. Note that in this case, since $(N_1, \boldsymbol{A}_1, \boldsymbol{M}_1, \delta_1)$ corresponds to (V, A) and $|\boldsymbol{M}_1| = 0$, we must have $V = N_1$. Likewise, since $(N_2, \boldsymbol{A}_2, \boldsymbol{M}_2, \delta_2)$ corresponds to (V, A) and $|\boldsymbol{M}_2| = 0$, we must have $V = N_2$. In other words, $N_1 = N_2$. Let $d_1 = 0$ if $\delta_1 = 0$, and $d_1 = 4$ otherwise. Define d_2 for δ_2 analogously. Since $(N_1, \boldsymbol{A}_1, \boldsymbol{M}_1, \delta_1)$ corresponds to (V, A) and $|\boldsymbol{M}_1| = 0$, we have $A = \langle \boldsymbol{A}_1[1], d_1\rangle$. Likewise, since $(N_2, \boldsymbol{A}_2, \boldsymbol{M}_2, \delta_2)$ corresponds to (V, A) and $|\boldsymbol{M}_2| = 0$, we have $A = \langle \boldsymbol{A}_2[1], d_2\rangle$. In other words, $\boldsymbol{A}_1 = \boldsymbol{A}_2$, and $\delta_1 = \delta_2$. Hence the two tuples $(N_1, \boldsymbol{A}_1, \boldsymbol{M}_1, \delta_1)$ and $(N_2, \boldsymbol{A}_2, \boldsymbol{M}_2, \delta_2)$ are the same, which is a contradiction.

$\square$

7.6.1 Weakened OAE2

In the original publication, we also proposed two variants of the OAE2 notion that are weaker with respect to nonce-reuse. The first, called nOAE, is essentially a purely nonce-based version of OAE2. The second, called dOAE, is halfway between nOAE and OAE2, in that it allows an adversary to repeat the nonce in order to extend an encryption chain that has already been finalized; this notion resembles the the security definition used by Bertoni et al. [BDPA11a, BDPV12], and we included it as a nod to the Keccak team, acknowledging that they set off in the right direction. Finally, we also provided a very natural nOAE-secure construction called STREAM.

Chapter 8

Authenticated Encryption with Variable Stretch

In this chapter, we address the problem of securely using nonce-based AE schemes with *variable-length tags* (variable stretch) under the same key.

The work presented in this chapter is a result of joint work with Reza Reyhanitabar and Serge Vaudenay which was published in ASIACRYPT 2016 [RVV16].

Organization of the Chapter. We first discuss related work in Section 8.1 and list the contributions in Section 8.2.

We then discuss the problem we study in more detail in Section 8.3 and we describe attacks that illustrate the dangers of stretch misuse in Section 8.4.

In Section 8.5, we proceed to formalizing the security of nonce-based AE schemes when used with variable stretch. We also establish relations with existing notions, and explain how to interpret the security bounds in our model.

Finally, we demonstrate how to construct efficient AE schemes that are secure in the sense of our new notion in Section 8.6.

8.1 Related Work

The use of a nonce-based AE scheme with varying amount of stretch per key can be seen as a misuse of the scheme, and thus our security notions can be interpreted as capturing resistance to this particular type of misuse. Other existing notions treat resistance of AE against misuse: MRAE by Rogaway and Shrimpton [RS06b], online nonce-misuse resistant AE by Fleischmann et al. [FFL12], AE under the release of unverified plaintext (AE-RUP) by Andreeva et al. [ABL$^+$14a], robust AE (RAE) by Hoang et al. [HKR15], online AE (OAE2) by Hoang et al. [HRRV15], and the notions that consider leakage by misuse of AE in protocols by Barwell et al. [BMOS17, BPS15]. None of these notions captures what should be the security of AE when one varies the stretch with the same key, except for RAE.

RAE aims to capture the "*best-possible*" AE security [HKR15]. Similar to the MRAE and Pseudorandom Injection (PRI) notions [RS06b], it targets robustness to nonce-misuse, but also robustness to decryption misuse and the security with variable stretch. However, the cost to pay for achieving such a strong goal is that any RAE scheme incurs a particular inefficiency: neither encryption nor decryption can be online. Our goal is different; rather than aiming at the best-possible security, we provide an enhancement to the popular NAE model that only adds robustness to tag-length variation under the same key, without sacrificing desirable features, such as onlineness of encryption.

The possibility of AE schemes' misbehaviour due to the use of variable-length tags has been discussed before. A thread in the CFRG forum discusses this for OCB [Man], and another discussion thread exists in the CAESAR competition mailing list [Iwa15].

Several schemes received ad-hoc measures that attempt to make their ciphertexts tag-dependent [KR14, Iwa15, Min15, Rey].

The attack presented in Section 8.4.2 is a generalization of the tag-length misusing attack on OMD (see Section 3.3) by the Ascon team [DEMS].

8.2 Contribution

We discuss the trivial security issues that arise from varying the stretch with nonce-based AE schemes and describe a stretch-misusing forgery attack that applies to a large class of nonce-based AE schemes, even with certain ad-hoc measures in place.

We define the security of nonce-based AE schemes with variable stretch through the notion NVAE. We establish the relations between NVAE, other notions defined in this chapter and previously existing notions. We show that, surprisingly, when variable tags are allowed, the all-in-one and the two-requirement security definitions are no longer equivalent. We additionally define the KESS notion as a useful, albeit strong, property that facilitates modular security proofs of NVAE security for AE schemes whose NAE security has already been established.

We demonstrate the feasibility of NVAE-security by designing OCBv, a variant of the well-known AE scheme OCB [RBBK01, Rog04a, KR11]. The modifications that transform OCB into OCBv and the security analysis are generic enough to be applied to other schemes based on tweakable blockciphers, or other tweakable primitives (e.g. compression functions), which represents a large subset of current nAE schemes.

8.3 Why Consider AE with Variable Stretch

Providing *authenticity* requires any AE scheme to grow the ciphertexts by a non-zero amount of stretch (see Section 2.4 for definition of stretch). All the known security notions for AE schemes [Rog02, Rog04b, RS06b, FFL12, ABL^{+}14a, HRRV15] and constructions thereof, with the exception of RAE [HKR15], assume that the stretch τ is a constant, or a parameter of the scheme which must be *fixed* per key, and security is proved under this assumption. A correct usage of such a scheme shall ensure that two instances of the same scheme with different stretches τ_1 and τ_2 always use two independently chosen keys K_1 and K_2. However, this rigid correct-use mandate may be violated in practice for various reasons.

Misuse. AE schemes may be used with variable-length tags per key due to misuse and poorly engineered security systems. With the increasing scale of deployment of cryptography, various types of misuse of cryptographic tools (i.e. their improper use that leads to compromised security) occur routinely in practice [LZLG14, EBFK13, Hot10, Wu05, BGW01, Lan14]. Identifying potential ways of misuse and mitigating their impact by sound design is therefore of great importance, while waving such a potential misuse off, because there have been no cases of occurrence yet, is a dangerous practice. Prior "Disasters" [Ber14b] have shown that it's a question of when, not if, a misuse will eventually happen in applications of (symmetric-key) cryptographic schemes.

The ongoing CAESAR competition [Ber14a] has explicitly listed a set of conventional confidentiality and integrity goals for AE, but has left "any additional security goals and robustness goals that the submitters wish to point out" as an option. Among the potential additional goals, *robustness* features (and in particular, different flavours of misuse-resistance to nonce reuse [RS06b, FFL12]) have attracted a lot of attention. While the recent focus has been mainly on nonce misuse, proper characterization and formalization of other potential misuse dimensions seems yet a challenge to be further investigated. The current literature lacks a systematic approach to formalizing an appropriate notion of AE with misuse-resistance to tag-length variation under the same key, *without sacrificing* interesting functional and efficiency features, such as onlineness of encryption.

Efficiency Constraints. Second, there are use cases, such as resource-constrained communication devices, where the support for variable-length tags is desired, but changing the key per tag length and renegotiating the system parameters is a costly process due to bandwidth and energy constraints. In those cases, supporting variable stretch per key while still being able to provide a "sliding scale" authenticity is deemed to be a useful feature, as pointed out by Struik [Str]. For instance, de Meulenaer et al. demonstrate that in the case of wireless sensor networks, communication-related energy consumption is substantially higher than the consumption caused by computation [dMGSP08]. Sliding scale authenticity could significantly extend the lifetime of such sensors, especially if processed plaintexts are very short, while only a handful of them requires a very high level of authenticity.

Discussions and measures. The problem has appeared to be highly interesting from both theoretical and practical perspectives as evidenced by the relatively long CFRG forum thread on issues arising from variable-length tags in OCB [Man], followed by ongoing discussions in the CAESAR competition mailing list [Iwa15], which in turn has motivated several second-round CAESAR candidates to be tweaked [Iwa15, Min15, Rey] with the aim of providing some *heuristic* measures for addressing the problem.

Terminology. Most standard AE schemes adopt a syntax in which the ciphertex is explicitly partitioned as $C = C_{\texttt{core}}||\texttt{Tag}$ with $C_{\texttt{core}}$ as the ciphertext core (decryptable to a putative plaintext) and $\texttt{Tag}$ as the authentication tag (used for verifying the decrypted message). In this chapter, we will use the terms *ciphertext expansion*, *stretch* and *tag length* interchangeably unless the syntax of an AE scheme (e.g. an RAE scheme) does not allow partitioning of the ciphertext to a core part and a tag part, in which case we use the general term *stretch*.

8.4 The Dangers of Stretch Misuse

Lack of support for variable-length tags per key in conventional AE security models, in particular in the popular NAE security model, is not just a theoretical and definitional complaint. All known standard AE schemes such as the widely-deployed CCM [Dwo04, WHF03a], GCM [MV04, Dwo07], and OCB do *misbehave* in one way or another if misused in this way [RW03, Man, Rog13]. Depending on the application scenario, the consequences of such a misbehavior may range from a degraded security level to a complete collapse.

8.4.1 Trivial Issues and Heuristic Fixes

A CFRG forum discussion thread initiated by Manger [Man] has raised concerns with an "Attacker changing tag length in OCB". While the discussion deals with issues identified for OCB, they apply to all nonce-based AE schemes $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ whose encryption algorithm outputs a core ciphertext and an authentication tag $\mathcal{E}(K, N, A, M) = C\|T$ such that $C = f_1(K, N, A, M)$ and $T = \mathsf{left}_\tau\,(f_2(K, N, A, M))$ for some functions f_1 and f_2, and a parameter $\tau \in \mathbb{N}$ (note that the output of f_2 is truncated to τ bits). The discussion can be summarized as follows:

- Assume that different instances of Π with several different tag lengths are defined and used. Under the same key, shorter tags are simply a truncation of longer tags. The tag length is clearly not mixed into the ciphertext. Consequently, given a valid output $C\|T = \mathcal{E}(K, N, A, M)$ with e.g. 128-bit tag, it is trivial to produce a valid output $C\|\mathsf{left}_{64}\,(T)$ for an instance of Π with 64-bit tags under the same key, by just dropping the last 8 bytes.

- An attacker wanting to change the associated data A to A' (e.g. from saying "TOP SECRET" to "PUBLIC") while keeping the same plaintext M as encrypted by the originator for a ciphertext $C\|T = \mathcal{E}(K, N, A, M)$ (with $|T|$ being e.g. 128 bits) only has to defeat the shortest accepted tag length (e.g. 64 bits) by trying to forge with $N, A, C\|T'$ with every possible T' (of 64 bits). This only applies if AD is not used when computing core ciphertext, i.e. if $f_1(K, N, A, M) = f'(K, N, M)$ for some function f'. It is the case in OCB, GCM, or OMD.

- Would it be better if the instances of the same algorithm with different tag lengths could not affect each other?

Heuristic Measures. The CFRG discussions concluded when the designers of OCB adopted the heuristic measure proposed by Manger: “just drop the tag length into the nonce” [Rog13]. For example, if the original nonce space $\mathcal{N} = \{0,1\}^\nu$, one may use an effective nonce N' of $\nu - t$ bits and encode τ in t bits, so that the encryption has the structure $\mathcal{E}(K, N' \| \langle\tau\rangle_t, A, M)$. One may call this method *nonce stealing* for tag length akin to nonce stealing for associated data (AD), proposed by Rogaway [Rog02] to convert a message-only AE scheme to an AE scheme with AD.

In a recent CAESAR competition discussion, Nandi [Nan] has raised the question whether including the tag length in the associated data can resolve the problem. For example, we may simply encode τ in t bits and prepend it to the AD, so that we encrypt as in $\mathcal{E}(K, N, \langle\tau\rangle_t \| A, M)$.

A natural extension would then be to combine both the measures, i.e., including the tag length as part of both the nonce and the associated data. But in the absence of a definitional and provable-security treatment, the proposed heuristic measures and claims for added security in the tweaked schemes are informal, and only limited to preventing some specific type of misbehavior by the schemes.

8.4.2 Failure of Inserting Stretch into Nonce and/or AD

Intuitively, one might hope that an AE scheme will guarantee τ_c-bit authenticity to the recipient whenever a received ciphertext with a τ_c-bit tag is decrypted, irrespective of the parallel existence other instances of the same algorithm, using the same key but different (shorter or longer) τ-bit tags. With the two heuristic measures described in Section 8.4.1 in place, one might even *expect* this intuition to be true.

Using a generic forgery attack, we show that the said heuristic measures fail to deliver on this expectation when applied to a large class of NAE schemes, including e.g. GCM and OCB. The class of schemes in question consists of all AE schemes that follow the “ciphertext translation” design paradigm of Rogaway [Rog02].

Ciphertext translation method transforms a message-only nonce-based AE scheme $\bar{\Pi} = (\bar{\mathcal{K}}, \bar{\mathcal{E}}, \bar{\mathcal{D}})$ with stretch τ and a keyed function $H : \mathcal{K}' \times \{0,1\}^* \to \{0,1\}^n$ with $n \geq \tau$ into a general nonce-based AE scheme $\Pi = (\bar{\mathcal{K}} \times \mathcal{K}', \mathcal{E}, \mathcal{D})$. A message-only AE scheme is one that does not accept any AD input, i.e. it has $\bar{\mathcal{A}} = \emptyset$. The ciphertext translation is defined in Figure 8.1.

The attack. We present a stretch-misusing forgery attack. We call it VTag-Forgery. It is a generalization of the tag-length misusing attack on OMD version 1 proposed by the Ascon team [DEMS]. OMD also follows the ciphertext translation method.

```
algo $\mathcal{E}_{\bar{K},K'}(N,A,M)$ :
    $C_M \leftarrow \bar{\mathcal{E}}_{\bar{K}}(N,M)$
    $T_A \leftarrow \mathsf{left}_\tau\,(H_{K'}(A))$
    $C \leftarrow C_M \oplus 0^{|M|}\|T_A$
    return $C$

algo $\mathcal{D}_{\bar{K},K'}(N,A,C)$ :
    $T_A \leftarrow \mathsf{left}_\tau\,(H_{K'}(A))$
    $C_M \leftarrow C \oplus 0^{|M|}\|T_A$
    $M \leftarrow \bar{\mathcal{D}}_{\bar{K}}(N,C_M)$
    return $M$
```

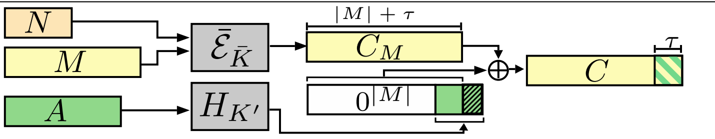


Figure 8.1 – **Ciphertext translation.** We construct a general nonce-based AE scheme $\Pi = (\bar{\mathcal{K}} \times \mathcal{K}', \mathcal{E}, \mathcal{D})$ with stretch τ from a message-only nonce-based AE scheme $\bar{\Pi} = (\bar{\mathcal{K}}, \bar{\mathcal{E}}, \bar{\mathcal{D}})$ also with stretch τ and a keyed function $H : \mathcal{K}' \times \{0,1\}^* \to \{0,1\}^n$ with $n \geq \tau$. The encryption and decryption algorithms $\mathcal{E}$ and $\mathcal{D}$ are defined on the top, the encryption illustrated on the bottom.

We target any nonce-based AE scheme Π constructed with ciphertext translation and parameterized by stretch τ. We assume that there are instances of Π defined with each amount of stretch from a set $\mathcal{I}_T = \{\tau_1, \ldots, \tau_r\}$ with $\tau_1 < \tau_2 < \ldots < \tau_r$.

We assume the adversary has oracle access to encryption and decryption algorithms, such that the amount of stretch can be chosen for every query independently. Note that this is equivalent to r co-existing instances of Π sharing the same key. The goal of the adversary is to forge a ciphertext for a given AD-message pair (A, M) expanded by $\tau_g \in \mathcal{I}_T$ bits, with $g > 1$.

To find the forgery, the adversary first makes an encryption query with M, arbitrary nonce N_1 and some arbitrary $A^* \neq A$ to the instance with the *shortest* stretch τ_1, and consequently finds a forgery for N_1, A and M with 2^{τ_1} decryption queries. Because of the ciphertext translation structure, all of the ciphertext, except the last $\tau - 1$ bits, will be the same. From the forgery and the encryption query, the adversary will be able to learn the first τ_1 bits of the *xor-difference* of $H_K(A)$ and $H_K(A^*)$.

Then, the adversary proceeds in a similar way with the second-shortest stretch τ_2, except when making the forgery for N_2, A, and M, it already knows the first τ_1 bits of $H_K(A) \oplus H_K(A^*)$, and only needs to guess the remaining $\tau_2 - \tau_1$, which means at most $2^{\tau_2 - \tau_1}$ decryption queries. The attacker then continues in a similar manner, making $2^{\tau_i - \tau_{i-1}}$ decryption queries with τ_i bits of stretch for $i = 1, \ldots, \tau_g$. An algorithmic description of the attack is given in Figure 8.2.

Applicability. With no measures for mixing the stretch in the encryption in place, the attack always succeeds. The keyed function $H_K(\cdot)$ must fulfil some mild conditions for the attack to work against the described heuristic countermeasures [Rog13, Nan], namely:

```
1:  algorithm VTag-Forgery($\tau_g, A, M$)
2:      $\Delta_A \leftarrow \varepsilon$; $A^* \leftarrow_\$ \mathcal{A}\backslash\{A\}$
3:      for $i \leftarrow 1$ to $g$ do
4:          pick fresh nonce $N_i$
5:          $C_i^* \leftarrow \mathrm{Enc}(N_i, A^*, \tau_i, M)$
6:          do
7:              pick fresh $\delta \in \{0,1\}^{\tau_i - \tau_{i-1}}$
8:              $C_i \leftarrow C_i^* \oplus 0^{|C_i| - \tau_i} \| \Delta_A \| \delta$
9:              $M_i \leftarrow \mathrm{Dec}(N_i, A, \tau_i, C_i)$
10:         while $M_i = \bot$
11:         $\Delta_A \leftarrow \mathsf{right}_{\tau_i}(C_i \oplus C_i^*)$
12:     end for
13:     return $N_g, A, C_g$
14: end algorithm
```

Figure 8.2 – **Ciphertext forgery** for a nonce-based AE scheme constructed with ciphertext translation with associated data A and message M in presence of variable stretch. Here $\tau_0 = 0$.

- If the stretch is only encoded in the nonce, the attack works with *arbitrary* $H_K(\cdot)$.

- For inclusion of the tag length in the AD or a combination of this method and nonce stealing, the attack works if two conditions are met. First, if the keyed function H can be described as $H_K(A) = H_{1_K}(A_1) \oplus H_{2_K}(A_2) \oplus \cdots \oplus H_{m_K}(A_m)$, for arbitrary functions $H_{i_K} : \{0,1\}^\mu \rightarrow \{0,1\}^n$, $1 \leq i \leq m$, where $A = A_1 \| A_2 \| \cdots \| A_m$ for $A_j \in \{0,1\}^\mu$ for some positive integer μ. We note that this is the case for both GCM[1] and OCB. Second, the value of stretch τ must only influence one μ-bit block of A (or a limited number thereof). This is the case if, for example, we prepend the AD with an encoding of τ as in $\mathcal{E}(K, N, \langle\tau\rangle_t \| A, M)$, and if at the same time $t \leq \mu$.
  Then the attack works, but we always pick A^*, so that the block(s) of AD that are actually affected by τ are the same in the queries on lines 5 and 9 in Figure 8.2.

In either of the two cases the attack will succeed: whenever we encrypt a message M with two different associated data A, A^*, first with τ_i and then with $\tau_j > \tau_i$ bits of stretch, then $C_i \oplus C_i^*$ will be a prefix of $C_j \oplus C_j^*$, as the xor cancels out the core ciphertext, and if the hash H is implemented as an xor-sum of sub-functions H_i then the block(s) of AD that are impacted by τ will cancel out as well.

Complexity. The complexity of the attack in terms of decryption queries will be $2^{\tau_1} + \sum_{i=2}^{g} 2^{\tau_i - \tau_{i-1}}$, which is dominated by 2^α with $\alpha = \max\{\tau_1, \tau_2 - \tau_1, \ldots, \tau_g - \tau_{g-1}\}$. For example, an adversary having access to four instances of an AE encryption algorithm with 32-bit, 64-bit, 96-bit and 128-bit tags under the same key will only need a total decryption query complexity $4 \cdot 2^{32}$ to forge a message with a 128-bit tag, which is in stark contrast with the 2^{128} decryption queries *expected* to be necessary to forge a ciphertext with 128 bits of stretch.

[1]Although the authentication tag is computed directly using AD and the ciphertext, each monomial evaluated in the polynomial hash used in GCM can be seen as H_{i_K}

8.5 Formalizing Nonce-based AE with Variable Stretch

Our goal in this section is to formalize the security of nonce-based AE schemes with variable stretch, or else their security in presence of stretch misuse. This turns out to be a non-trivial task, as evidenced by the previously mentioned discussions [Man, Rog13, RW03].

Allowing the adversary to choose the amount of stretch freely from a set $\mathcal{I}_T = \{\tau_{\min}, \ldots, \tau_{\max}\}$ will inevitably enable it to produce forgeries with a high probability $2^{-\tau_{\min}}$, by targeting the shortest allowed stretch; a forgery is sure to be found with at most $2^{\tau_{\min}}$ verification queries. This is inherent to any AE scheme.

Despite this limit to its *global* security guarantees, there is a meaningful security property which can be expected from an variable-stretch AE scheme by a user: the scheme must guarantee τ bits of security for ciphertexts with τ bits of stretch, regardless of adversarial access to other instances with the same key but other (shorter and/or longer) amount of stretch than τ. For example, forging a ciphertext with τ-bit stretch should require $\approx 2^{\tau}$ verification queries *with τ-bit stretch*, regardless of the number of queries made with different amounts of stretch.

This non-interference between different instances that use the same key but different stretch (tag length) is the intuition that we capture in our security notion for nonce-based, variable-stretch AE schemes.

Syntax. We augment the syntax of nonce-based AE schemes to include an input that allows to control the amount of stretch applied upon every encryption individually. We will call such augmented AE schemes *variable-stretch* AE schemes.

A variable-stretch AE scheme is a triplet $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ where the key space $\mathcal{K}$ is a set endowed with a probabilistic distribution, and $\mathcal{E} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{I}_T \times \mathcal{M} \to \mathcal{C}$ and $\mathcal{D} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathbb{N} \times \mathcal{C} \to \mathcal{M} \cup \{\perp\}$ are the encryption and decryption algorithm respectively, both deterministic and stateless. The nonce space $\mathcal{N}$, AD space $\mathcal{A}$, plaintext space $\mathcal{M}$, and ciphertext space $\mathcal{C}$ are all subsets of $\{0,1\}^*$. We call the finite set $\mathcal{I}_T$ stretch space of Π (i.e. the set of ciphertext expansion values that can be applied upon encryption), and we require that $\mathcal{I}_T \subseteq \mathbb{N}$.

We require for every $M \in \{0,1\}^*$ that if $M \in \mathcal{M}$ then $M' \in \mathcal{M}$ for all $M' \in \{0,1\}^{|M|}$. We further require that any variable-stretch scheme is *correct*; for every $(K, N, A, \tau, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{I}_T \times \mathcal{M}$, we require that if $\mathcal{E}(K, N, A, \tau, M) = C$ then $\mathcal{D}(K, N, A, \tau, C) = M$. Finally, we require that the scheme applies the requried stretch, i.e. for every $(K, N, A, \tau, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{I}_T \times \mathcal{M}$, we require that for $\mathcal{E}(K, N, A, \tau, M) = C$ we have $|C| = |M| + \tau$.

The variable-stretch AE syntax is easily seen to be an extension of the nonce-based AE syntax: an instance of the conventional nonce-based AE scheme Π with the ciphertext expansion fixed to some constant value τ is equivalent to modelling Π as a variable-stretch AE scheme and setting $\mathcal{I}_T = \{\tau\}$. We sometimes create an ordinary nonce-based AE scheme Π' from a nonce-based AE scheme with variable stretch Π by fixing the expansion value for all queries to some value $\tau \in \mathcal{I}_T$. We will denote this as $\Pi' = \Pi[\tau]$.

8.5.1 NVAE Security

We define a new security notion called NVAE as an extension to the all-in-one definition of NAE security. In contrast to NAE, the new security games and the advantage function of NVAE are parameterized by the challenge stretch value $\tau_c \in \mathcal{I}_T$, which models the intentions of the adversary to target the ciphertexts treated with precisely τ_c bits of stretch. We first describe the security model informally, before proceeding to the formal definition.

Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a variable-stretch AE scheme. An NVAE adversary $\mathscr{A}$ gets to interact with games $\mathbf{nvae\text{-}R}(\tau_c)_\Pi$ (left) and $\mathbf{nvae\text{-}I}(\tau_c)_\Pi$ (right) in Figure 8.3, defining respectively the real and ideal behaviour of such a scheme. The adversary has access to two oracles Enc and Dec determined by these games and its goal is to distinguish the two games. We stress that these oracles now allow to choose the stretch.

The adversary must respect a *relaxed nonce-requirement*; it must use a unique pair of nonce and stretch for each encryption query. Compared to the standard nonce-respecting requirement in NAE schemes, here a nonce may be reused, provided that the stretch does not repeat simultaneously.

In the ideal game $\mathbf{nvae\text{-}I}(\tau_c)_\Pi$, *only* the encryption and decryption queries with τ_c-bit stretch are answered in the same idealized way as in the "ideal" game of NAE notion (Figure 2.2). The queries with stretch other than τ_c are treated with the real encryption (or decryption) algorithm. This lets the adversary to issue arbitrary queries (e.g. repeated forgeries) for any stretch $\tau \neq \tau_c$ and leverage the information thus gathered to attack the challenge expansion. At the same time, only queries with τ_c bits of stretch can help the adversary to actually distinguish the two games, capturing the exact level of security for ciphertexts treated with τ_c bits of stretch, in presence of variable stretch. The NVAE security is formally stated in Definition 8.1.

Adversarial resources. The adversarial resources of interest for the **nvae** notion are $(t, \mathbf{q_e}, \mathbf{q_d}, \boldsymbol{\sigma})$, where t denotes the running time of the adversary, $\mathbf{q_e} = (\boldsymbol{q_e}[\tau] | \tau \in \mathcal{I}_T)$ denotes an array indexed by stretch that holds the number of encryption queries $\boldsymbol{q_e}[\tau]$ made with stretch τ for every stretch $\tau \in \mathcal{I}_T$,' $\mathbf{q_d} = (\boldsymbol{q_d}[\tau] | \tau \in \mathcal{I}_T)$ denotes the same for the decryption queries, and $\boldsymbol{\sigma} = (\boldsymbol{\sigma}[\tau] | \tau \in \mathcal{I}_T)$ denotes the array that holds the total amount of data $\boldsymbol{\sigma}[\tau]$ processed in all queries with stretch τ for every $\tau \in \mathcal{I}_T$.

Despite being focused on queries stretched by τ_c bits, we watch adversarial resources for every stretch $\tau \in \mathcal{I}_T$ in a detailed, vector-based fashion. This approach appears to be most versatile with respect to the security analysis. However, in a typical case, we will be interested in the resources related to τ_c (i.e. $\boldsymbol{q}_e[\tau_c], \mathbf{q_d}[\tau_c], \boldsymbol{\sigma}[\tau_c]$) and cumulative resources of the adversary q_e, q_d, σ with $q_e = \sum_{\tau \in \mathcal{I}_T} \boldsymbol{q_e}[\tau]$, $q_d = \sum_{\tau \in \mathcal{I}_T} \boldsymbol{q_d}[\tau]$ and $\sigma = \sum_{\tau \in \mathcal{I}_T} \boldsymbol{\sigma}[\tau]$.

Definition 8.1 (NVAE(τ_c) security)**.** *Given a variable stretch AE scheme* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *with a stretch space* $\mathcal{I}_T$*, a challenge amount of stretch* $\tau_c \in \mathcal{I}_T$ *and an adversary* $\mathscr{A}$*, we* *define the advantage of $\mathscr{A}$ in breaking the NVAE(τ_c) security of Π in a chosen ciphertext attack (with help of the games* **nvae-*R***(τ_c) *and* **nvae-*I***(τ_c) *in Figure 8.3) as*

$$\mathbf{Adv}_{\Pi}^{\mathbf{nvae}(\tau_c)}(\mathscr{A}) = \Pr[\mathscr{A}^{\mathbf{nvae}\text{-}\boldsymbol{R}(\tau_c)_\Pi} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{nvae}\text{-}\boldsymbol{I}(\tau_c)_\Pi} \Rightarrow 1].$$

If $\mathbf{Adv}_{\Pi}^{\mathbf{nvae}(\tau_c)}(\mathscr{A}) \le \epsilon$ for all adversaries $\mathscr{A}$ whose resources are limited by $(t, \boldsymbol{q}_e, \mathbf{q_d}, \boldsymbol{\sigma})$, then we say that Π is a $(\epsilon, \tau_c, t, \boldsymbol{q}_e, \mathbf{q_d}, \boldsymbol{\sigma})$-secure variable-stretch AE scheme.

General NVAE security. By parameterizing the definition of NVAE security by τ_c, we have only postponed the difficult question: how do we evaluate the *overall* security of a variable-stretch scheme? This is where we make use of the parametrization of the notion. We will use it to state a comprehensible, albeit informal definition of general NVAE security.

We note that a similar informal definition can be made for all further security notions parameterized by stretch that we introduce in this chapter.

Definition 8.2 (NVAE-security (informal))**.** *We say that a variable-stretch scheme Π is NVAE-secure if for every $\tau_c \in \mathcal{I}_T$, and for all adversaries $\mathscr{A}$ with "reasonable" resources $(t, \boldsymbol{q}_e, \mathbf{q_d}, \boldsymbol{\sigma})$, the advantage $\mathbf{Adv}_{\Pi}^{\mathbf{nvae}(\tau_c)}(\mathscr{A})$ is reasonably "small" with respect to τ_c.*

The key phrase in this informal definition is "with respect to τ_c": the $\mathbf{Adv}_{\Pi}^{\mathbf{nvae}(\tau_c)}(\mathscr{A})$ adversarial advantage will inevitably be lower-bounded by $\mathbf{q_d}[\tau_c]/2^{\tau_c}$, which will be big if τ_c is small. However, this is expected. What we want is that the τ_c-specific advantage never departs too far from this unavoidable lower bound. We further discuss the interpretation of the **nvae** bounds in Section 8.5.5.

Remark 8 (Relation to NAE)**.** *The notion of* **nvae** *is indeed an extension of the classical all-in-one security notion for nonce-based AE schemes. If for a variable-stretch AE scheme Π with a stretch-space $\mathcal{I}_T$ the advantage $\mathbf{Adv}_{\Pi}^{\mathbf{nvae}(\tau_c)}(\mathscr{A})$ is small for every reasonable $\mathscr{A}$ and for every $\tau_c \in \mathcal{I}_T$, then it will be small for any τ_c in a smaller stretch-space $\mathcal{I}'_T \subseteq \mathcal{I}_T$, including $\mathcal{I}'_T = \{\tau_c\}$. If a scheme has a trivial stretch-space $\mathcal{I}_T = \{\tau_c\}$, then NVAE becomes the classical NAE notion. It easily follows, that NVAE(τ_c) security of a scheme Π tightly implies security of $\Pi[\tau_c]$. In particular, $\mathbf{Adv}_{\Pi}^{\mathbf{nae}}(t, \boldsymbol{q}_e[\tau_c], \mathbf{q_d}[\tau_c], \boldsymbol{\sigma}[\tau_c]) \le \mathbf{Adv}_{\Pi[\tau_c]}^{\mathbf{nvae}(\tau_c)}(t', \mathbf{q_e}, \mathbf{q_d}, \boldsymbol{\sigma})$.*

Parameterized CCA security. An NAE-secure AE scheme is also IND-CCA-secure. This follows from the equivalence of the all-in-one and dual AE notions (Lemma 2.9) and a well-known implication PRIV $\wedge$ AUTH $\Rightarrow$ IND-CCA established by Bellare and Namprempre [BN00] (see the definition of IND-CCA security in Appendix B.4).[2] It is natural to ask: *Does the NVAE(τ_c)-security also provide a privacy guarantee against*

[2]Even though the result of Bellare and Namprempre applies to randomized schemes, simply citing it in the context of deterministic nonce-based schemes has become folklore. We commit the same act of negligence here, but we note that a very similar analysis to that of Bellare's and Namprempre's performed with the corresponding notions for nonce-based AE schemes would yield a very similar result.

```
proc initialize                    nvae-R(τ_c)_Π
$K \leftarrow\$ \mathcal{K}$
$\mathcal{X} \leftarrow \varnothing$, $\mathcal{Y} \leftarrow \varnothing$

oracle Enc$(N, A, \tau, M)$
if $(N, \tau) \in \mathcal{X}$ then
  return $\perp$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{(N, \tau)\}$
$C \leftarrow \mathcal{E}(K, N, A, \tau, M)$
if $\tau = \tau_c$ then
  $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(N, A, C)\}$
return $C$

oracle Dec$(N, A, \tau, C)$
if $\tau = \tau_c$ and $(N, A, C) \in \mathcal{Y}$ then
  return $\perp$
return $\mathcal{D}(K, N, A, \tau, C)$
```

```
proc initialize                    nvae-I(τ_c)_Π
$K \leftarrow\$ \mathcal{K}$
$\mathcal{X} \leftarrow \varnothing$

oracle Enc$(N, A, \tau, M)$
if $(N, \tau) \in \mathcal{X}$ then
    return $\perp$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{(N, \tau)\}$
if $\tau = \tau_c$ then
    $C \leftarrow\$ \{0,1\}^{|M|+\tau_c}$
    return $C$
return $\mathcal{E}(K, N, A, \tau, M)$

oracle Dec$(N, A, \tau, C)$
if $\tau = \tau_c$ then
    return $\perp$
return $\mathcal{D}(K, N, A, \tau, C)$
```

Figure 8.3 – **AE security with variable stretch.** Security games for defining AE security of a variable-stretch AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$.

chosen ciphertext attacks? We define IND-VCCA, an extension of the IND-CCA security notion for variable stretch AE schemes with τ_c-parameterized security games and answer this question positively.

The parameterized games of IND-VCCA notion capture the exact privacy level guaranteed by an variable-stretch AE scheme for encryption queries stretched by τ_c bits, in presence of arbitrary queries with expansions $\tau \neq \tau_c$ and non-trivial decryption queries stretched by τ_c bits. The notion builds on the intuition that privacy level of τ_c-expanded queries should not be affected by any adversarial queries made with other amounts of stretch.

Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a variable-stretch AE scheme. We let an adversary $\mathscr{A}$ interact with the games **ind-vcca-R**$(\tau_c)_\Pi$ and **ind-vcca-I**$(\tau_c)_\Pi$ defined in Figure 8.4 and its goal is to distinguish them. In the "ideal" game **ind-vcca-I**$(\tau_c)_\Pi$, the τ_c-stretched encryption queries are answered with random strings while the decryption queries are processed with the real decryption algorithm. $\mathscr{A}$ must respect the relaxed nonce-requirement and is prevented to win the game trivially (i.e. by re-encrypting output of decryption query with τ_c bits of stretch and vice-versa).

The adversarial resources of interest for the IND-VCCA notion are the same as for the NVAE notion, i.e. $(t, \mathbf{q_e}, \mathbf{q_d}, \boldsymbol{\sigma})$.

Definition 8.3 (IND-VCCA(τ_c) security)**.** *Given a variable stretch AE scheme* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *with a stretch space* $\mathcal{I}_T$*, a challenge amount of stretch* $\tau_c \in \mathcal{I}_T$ *and an adversary* $\mathscr{A}$*, we define the advantage of* $\mathscr{A}$ *in breaking the IND-VCCA*(τ_c) *security of* Π *in a* *chosen ciphertext attack (with help of the games* **ind-vcca-*R***(τ_c) *and* **ind-vcca-*I***(τ_c) *in Figure 8.4) as*

$$\mathbf{Adv}_{\Pi}^{\mathbf{ind\text{-}vcca}(\tau_c)}(\mathscr{A}) = \Pr\left[\mathscr{A}^{\mathbf{ind\text{-}vcca\text{-}}\boldsymbol{R}(\tau_c)_\Pi} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathbf{ind\text{-}vcca\text{-}}\boldsymbol{I}(\tau_c)_\Pi} \Rightarrow 1\right].$$

If $\mathbf{Adv}_{\Pi}^{\mathbf{ind\text{-}vcca}(\tau_c)}(\mathscr{A}) \leq \epsilon$ *for every adversary* $\mathscr{A}$ *with resources limited by* $(t, \boldsymbol{q}_e, \mathbf{q_d}, \boldsymbol{\sigma})$, *then we say that* Π *is* $(\epsilon, \tau_c, t, \boldsymbol{q}_e, \mathbf{q_d}, \boldsymbol{\sigma})$*-IND-VCCA secure.*

Remark 9 (Relations to IND-CCA and NVAE(τ_c))**.** *Similarly as in the case of NVAE and NAE, IND-VCCA security with some stretch space* $\mathcal{I}_T$ *implies IND-CCA security with any stretch space* $\mathcal{I}_T' \subseteq \mathcal{I}_T$*, among others* $\mathcal{I}_T = \{\tau_c\}$*. It follows that IND-VCCA*(τ_c) *security of a scheme* Π *implies the classical IND-CCA security of* $\Pi[\tau_c]$.

The notions IND-VCCA and NVAE differ in the way the "ideal" games treat the decryption queries expanded by τ_c *bits; the IND-VCCA notion does not capture integrity of ciphertexts. E.g. a scheme that concatenates output of a length-preserving, nonce-based, ind-cca-secure encryption scheme (using encoding of the nonce and stretch as a "nonce") and an image of the nonce and stretch under a PRF would be secure in the sense of IND-VCCA, but insecure in the sense of NVAE. Thus IND-VCCA*$(\tau_c) \not\Rightarrow$ *NVAE*(τ_c).

We formally treat the relation between the two notions in the opposite direction in Theorem 8.4. We would like to stress that the result in Theorem 8.4 holds for *any* variable-stretch AE scheme, and in particular for any stretch space $\mathcal{I}_T$.

Theorem 8.4 (NVAE(τ_c) $\Rightarrow$ IND-VCCA(τ_c))**.** *Let* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *be an arbitrary variable-stretch AE scheme, and* $\mathscr{A}$ *be an adversary with resources* $(t, \mathbf{q_e}, \mathbf{q_d}, \boldsymbol{\sigma})$*. We have that*

$$\mathbf{Adv}_{\Pi}^{\mathbf{ind\text{-}vcca}(\tau_c)}(\mathscr{A}) \leq 2 \cdot \mathbf{Adv}_{\Pi}^{\mathbf{nvae}(\tau_c)}(\mathscr{B}),$$

for some $\mathscr{B}$ *with resources* $(t', \mathbf{q_e}, \mathbf{q_d}, \boldsymbol{\sigma})$ *where* $t' = t + \gamma \cdot \left(\sum_{\tau \in \mathcal{I}_T} \boldsymbol{\sigma}[\tau]\right)$ *and* γ *is a constant.*

Proof. Let $\mathscr{A}$ be an IND-VCCA adversary with indicated resources. We define the game $\mathbf{ind\text{-}vcca\text{-}I}(\tau_c)_\Pi^\perp$ as an intermediate step in the proof; it is exactly the same as $\mathbf{ind\text{-}vcca\text{-}I}(\tau_c)_\Pi$, except that the decryption queries with τ_c bits of stretch are always answered with $\perp$. We have that

$$\begin{aligned}\mathbf{Adv}_{\Pi}^{\mathbf{ind\text{-}vcca}(\tau_c)}(\mathscr{A}) = {} & \Pr[\mathscr{A}^{\mathbf{ind\text{-}vcca\text{-}R}(\tau_c)_\Pi} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{ind\text{-}vcca\text{-}I}(\tau_c)_\Pi^\perp} \Rightarrow 1] \\ & + \Pr[\mathscr{A}^{\mathbf{ind\text{-}vcca\text{-}I}(\tau_c)_\Pi^\perp} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{ind\text{-}vcca\text{-}I}(\tau_c)_\Pi} \Rightarrow 1].\end{aligned}$$

We start by showing that

$$\Pr[\mathscr{A}^{\mathbf{ind\text{-}vcca\text{-}R}(\tau_c)_\Pi} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{ind\text{-}vcca\text{-}I}(\tau_c)_\Pi^\perp} \Rightarrow 1] \leq \mathbf{Adv}_{\Pi}^{\mathbf{nvae}(\tau_c)}(\mathscr{B})$$

for an NVAE adversary $\mathscr{B}$ with the resources $(t', \mathbf{q_e}, \mathbf{q_d}, \boldsymbol{\sigma})$. The reduction of $\mathscr{A}$ to $\mathscr{B}$ is straightforward: $\mathscr{B}$ simply answers $\mathscr{A}$'s queries with its own oracles, making sure that

```
proc initialize                          ind-vcca-R(τ_c)_Π
$K \leftarrow\!\$\ \mathcal{K}$
$\mathcal{V} \leftarrow \varnothing, \mathcal{X} \leftarrow \varnothing, \mathcal{Y} \leftarrow \varnothing$

oracle Enc$(N, A, \tau, M)$
if $(N, \tau) \in \mathcal{X}$ then return $\perp$
if $\tau = \tau_c$ and $(N, A, M) \in \mathcal{V}$ then
  return $\perp$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{(N, \tau)\}$
$C \leftarrow \mathcal{E}(K, N, A, \tau, M)$
if $\tau = \tau_c$ then
  $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(N, A, C)\}$
return $C$

oracle Dec$(N, A, \tau, C)$
if $\tau = \tau_c$ and $(N, A, C) \in \mathcal{Y}$ then
  return $\perp$
$M \leftarrow \mathcal{D}(K, N, A, \tau, C)$
if $\tau = \tau_c$ and $M \neq \perp$
  $\mathcal{V} \leftarrow \mathcal{V} \cup \{(N, A, M)\}$
return $M$
```

```
proc initialize                          ind-vcca-I(τ_c)_Π
$K \leftarrow\!\$\ \mathcal{K}$
$\mathcal{V} \leftarrow \varnothing, \mathcal{X} \leftarrow \varnothing, \mathcal{Y} \leftarrow \varnothing$

oracle Enc$(N, A, \tau, M)$
if $(N, \tau) \in \mathcal{X}$ then return $\perp$
if $\tau = \tau_c$ and $(N, A, M) \in \mathcal{V}$ then
    return $\perp$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{(N, \tau)\}$
if $\tau = \tau_c$ then
    $C \leftarrow\!\$\ \{0,1\}^{|M|+\tau_c}$
    $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(N, A, C)\}$
    return $C$
return $\mathcal{E}(K, N, A, \tau, M)$

oracle Dec$(N, A, \tau, C)$
if $\tau = \tau_c$ and $(N, A, C) \in \mathcal{Y}$ then
    return $\perp$
$M \leftarrow \mathcal{D}(K, N, A, \tau, C)$
if $\tau = \tau_c$ and $M \neq \perp$
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{(N, A, M)\}$
return $M$
```

Figure 8.4 – **Games for defining IND-VCCA security** of a variable-stretch AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$.

the trivial-win preventing restrictions of **ind-vcca** games are met. At the end of the experiment, $\mathscr{B}$ outputs whatever $\mathscr{A}$ outputs. This ensures perfect simulation of both games for $\mathscr{A}$.

It remains to show that

$$\Pr[\mathscr{A}^{\textbf{ind-vcca-I}(\tau_c)^{\perp}_{\Pi}} \Rightarrow 1] - \Pr[\mathscr{A}^{\textbf{ind-vcca-I}(\tau_c)_{\Pi}} \Rightarrow 1] \leq \mathbf{Adv}^{\mathbf{nvae}(\tau_c)}_{\Pi}(\mathscr{C})$$

for an NVAE adversary $\mathscr{C}$ with resources $(t', \mathbf{q_e}, \mathbf{q_d}, \boldsymbol{\sigma})$. We reduce $\mathscr{A}$ to $\mathscr{C}$ as follows. $\mathscr{C}$ answers all $\mathscr{A}$'s queries directly with its own oracles (again making sure to enforce all the restrictions of **ind-vcca** games), except for encryption queries expanded by τ_c bits. For those, $\mathscr{C}$ ignores its encryption oracle and answers with $|M| + \tau_c$ random bits if $\mathscr{A}$'s query has a fresh nonce-stretch pair and is not a re-encryption of the output of a previous decryption query. At the end of experiment, $\mathscr{C}$ outputs the inverse of $\mathscr{A}$'s output. If $\mathscr{C}$ interacts with **nvae-R**$(\tau_c)_\Pi$, then it perfectly simulates **ind-vcca-I**$(\tau_c)_\Pi$ for $\mathscr{A}$ while if $\mathscr{C}$ interacts with **nvae-I**$(\tau_c)_\Pi$, then it perfectly simulates **ind-vcca-I**$(\tau_c)^{\perp}_{\Pi}$. $\square$

```
proc initialize                         vpriv-R(τ_c)_Π
$K \leftarrow\!\!\$\ \mathcal{K}$
$\mathcal{X} \leftarrow \varnothing$

oracle Enc$(N, A, \tau, M)$
if $(N, \tau) \in \mathcal{X}$ then
  return $\perp$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{(N, \tau)\}$
return $\mathcal{E}(K, N, A, \tau, M)$
```

```
proc initialize                         vpriv-I(τ_c)_Π
$K \leftarrow\!\!\$\ \mathcal{K}$
$\mathcal{X} \leftarrow \varnothing$

oracle Enc$(N, A, \tau, M)$
if $(N, \tau) \in \mathcal{X}$ then
  return $\perp$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{(N, \tau)\}$
if $\tau = \tau_c$ then
  $C \leftarrow\!\!\$\ \{0,1\}^{|M|+\tau_c}$
  return $C$
return $\mathcal{E}(K, N, A, \tau, M)$
```

```
proc initialize                         vauth(τ_c)_Π
$K \leftarrow\!\!\$\ \mathcal{K}$
$\mathcal{X} \leftarrow \varnothing$, $\mathcal{Y} \leftarrow \varnothing$

oracle Enc$(N, A, \tau, M)$
if $(N, \tau) \in \mathcal{X}$ then
    return $\perp$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{(N, \tau)\}$
$C \leftarrow \mathcal{E}(K, N, A, \tau, M)$
if $\tau = \tau_c$ then
    $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(N, A, C)\}$
return $C$

oracle Dec$(N, A, \tau, C)$
if $\tau = \tau_c$ and $(N, A, C) \in \mathcal{Y}$ then
    return $\perp$
return $\mathcal{D}(K, N, A, \tau, C)$
```

Figure 8.5 – **Security games for defining VPRIV and VAUTH security of a variable-stretch AE scheme $\boldsymbol{\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})}$.**

8.5.2 No Two-Requirement Notion

The equivalence of the two-requirement security definition and the all-in-one security definition for AE security is among the best known results in the field [RS06b] (see Lemma 2.9). One may wonder whether such an equivalence also holds in the setting of variable-stretch AE schemes, for the natural τ_c-parameterized extensions of these notions. Surprisingly, we answer this question negatively. We consider the conventional confidentiality and authenticity notions for AE schemes [BN00, Rog02] and define their extensions with variable stretch, the notions VPRIV and VAUTH, as natural extensions of their conventional counterparts.

Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a variable-stretch AE scheme. An adversary $\mathscr{A}$ against confidentiality of Π interacts with games **vpriv-R**$(\tau_c)_\Pi$ (real scheme) and **vpriv-I**$(\tau_c)_\Pi$ (ideal behaviour) defined in Figure 8.5, both parameterized by the challenge stretch τ_c, and tries to distinguish them. An adversary $\mathscr{A}$ that attacks authenticity of Π with target stretch τ_c is left to interact with the game **vauth**$(\tau_c)_\Pi$ defined in Figure 8.5 and its goal is to find a valid forgery (i.e. produce a decryption query returning $M \neq \perp$) with the target stretch of τ_c bits.

Definition 8.5 (VPRIV(τ_c) and VAUTH(τ_c) security). *Given a variable stretch AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with a stretch space $\mathcal{I}_T$, a challenge amount of stretch $\tau_c \in \mathcal{I}_T$ and an adversary $\mathscr{A}$, we define the advantage of $\mathscr{A}$ in breaking the VPRIV(τ_c) security of Π in a chosen plaintext attack (with help of the games* **vpriv-*R***(τ_c) *and* **vpriv-*I***(τ_c) *in Figure 8.5) as*

$$\mathbf{Adv}_{\Pi}^{\mathbf{vpriv}(\tau_c)}(\mathscr{A}) = \Pr[\mathscr{A}^{\mathbf{vpriv}\text{-}\boldsymbol{R}(\tau_c)_\Pi} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{vpriv}\text{-}\boldsymbol{I}(\tau_c)_\Pi} \Rightarrow 1].$$

Given an adversary $\mathscr{A}'$, we define the advantage of $\mathscr{A}$ in breaking the VAUTH(τ_c) security of Π in a chosen ciphertext attack (with help of the game **vauth**(τ_c) *in Figure 8.5) as*

$$\mathbf{Adv}_{\Pi}^{\mathbf{vauth}(\tau_c)}(\mathscr{A}') = \Pr\left[\mathscr{A}'^{\mathbf{vauth}(\tau_c)_\Pi} \text{ forges with } \tau_c\right].$$

where $\mathscr{A}$ forges *denotes the event that the* Dec *oracle returns a value different from $\perp$.*

If $\mathbf{Adv}_{\Pi}^{\mathbf{vpriv}(\tau_c)}(\mathscr{A}) \leq \epsilon$ for all adversaries $\mathscr{A}$ whose resources are limited by $(t, \boldsymbol{q}_e, \boldsymbol{\sigma})$, then we say that Π is $(\epsilon, \tau_c, t, \boldsymbol{q}_e, \boldsymbol{\sigma})$-VPRIV secure. If $\mathbf{Adv}_{\Pi}^{\mathbf{vauth}(\tau_c)}(\mathscr{A}') \leq \epsilon'$ for all adversaries $\mathscr{A}'$ whose resources are limited by $(t', \boldsymbol{q}'_e, \boldsymbol{q}'_d, \boldsymbol{\sigma}')$, then we say that Π is $(\epsilon', \tau_c, t', \boldsymbol{q}'_e, \boldsymbol{q}'_d, \boldsymbol{\sigma}')$-VAUTH secure.

Remark 10 (Relations with NVAE, PRIV and AUTH notions). *As before, if a scheme Π is VPRIV(τ_c) (respectively VAUTH(τ_c)) secure with stretch-space $\mathcal{I}_T$, then it will be secure for any stretch-space $\mathcal{I}'_T \subseteq \mathcal{I}_T$ including $\mathcal{I}'_T = \{\tau_c\}$, implying the PRIV (respectively AUTH) security of the scheme $\Pi[\tau_c]$.*

We can easily verify that the NVAE(τ_c) security of a scheme Π implies both the VPRIV(τ_c) security and the VAUTH(τ_c) of Π, by adapting the reductions for corresponding conventional notions [RS06b] slightly. In Proposition 8.6, we show that the converse of this implication does not hold.

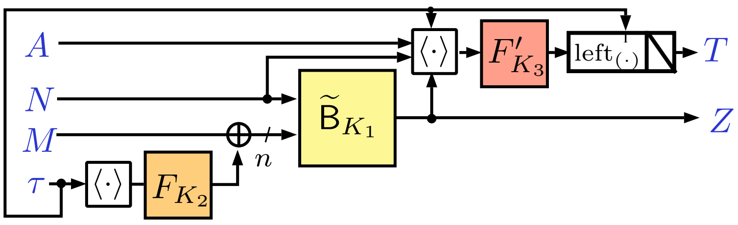


Figure 8.6 – **The encryption algorithm of the scheme $\Pi_{\neg\mathbf{cca}}$.** Here $\langle\cdot\rangle$ is an efficiently computable, injective encoding scheme.

Proposition 8.6. *Assuming the existence of secure tweakable blockciphers and PRFs, there exists a variable-stretch AE scheme, that is secure in the sense of the VPRIV(τ_c) notion and the VAUTH(τ_c) notion but insecure in the sense of IND-VCCA(τ_c) notion,* *i.e.*

$$VPRIV(\tau_c) \wedge VAUTH(\tau_c) \not\Rightarrow IND\text{-}VCCA(\tau_c).$$

Corollary 8.7. *There exists a variable-stretch AE scheme, that is secure in the sense of both the $VPRIV(\tau_c)$ notion and the $VAUTH(\tau_c)$ notion but insecure in the sense of $NVAE(\tau_c)$ notion, i.e.*

$$VPRIV(\tau_c) \wedge VAUTH(\tau_c) \not\Rightarrow NVAE(\tau_c).$$

Proof. To support the claim in Proposition 8.6, we define the variable-stretch AE scheme $\Pi_{\neg\text{cca}} = (\mathcal{K}_{\neg\text{cca}}, \mathcal{E}_{\neg\text{cca}}, \mathcal{D}_{\neg\text{cca}})$ constructed from a TPRP-secure tweakable blockcipher $\widetilde{E} : \mathcal{K}_1 \times \mathcal{N} \times \{0,1\}^n \to \{0,1\}^n$ and two PRF-secure keyed functions $F : \mathcal{K}_2 \times \{0,1\}^* \to \{0,1\}^n$ and $F' : \mathcal{K}_3 \times \{0,1\}^* \to \{0,1\}^m$. We define $\mathcal{K}_{\neg\text{cca}} = \mathcal{K}_1 \times \mathcal{K}_2 \times \mathcal{K}_3$, $\mathcal{M}_{\neg\text{cca}} = \{0,1\}^n$, $\mathcal{A}_{\neg\text{cca}} = \{0,1\}^*$, $\mathcal{N}_{\neg\text{cca}} = \mathcal{N}$ and the encryption and decryption algorithms as in Figure 8.7. We require that $|\mathcal{I}_{T\neg\text{cca}}| \geq 2$ and that $m \geq \max(\mathcal{I}_{T\neg\text{cca}})$. The encryption algorithm $\mathcal{E}_{\neg\text{cca}}$ is illustrated in Figure 8.6.

```
algo $\mathcal{E}_{\neg cca}(K, N, A, \tau, M)$
Parse $K$ as $K_1, K_2, K_3$
$W \leftarrow M \oplus F(K_2, \langle\tau\rangle)$
$Z \leftarrow \widetilde{E}(K_1, N, W)$
$T \leftarrow \mathsf{left}_\tau\,(F'(K_3, \langle N, A, \tau, Z\rangle))$
return $Z \| T$
```

```
algo $\mathcal{D}_{\neg cca}(K, N, A, \tau, C)$
Parse $K$ as $K_1, K_2, K_3$
Parse $C$ as $Z \| T$ with $|T| = \tau$
if $\mathsf{left}_\tau\,(F'(K_3, \langle N, A, \tau, Z\rangle)) \neq T$ then
    return $\bot$
$W \leftarrow \widetilde{E}^{-1}(K_1, N, Z)$
return $W \oplus F(K_2, \langle\tau\rangle)$
```

Figure 8.7 – **Encryption and decryption algorithms of the variable-stretch AE scheme $\Pi_{\neg\text{cca}} = (\mathcal{K}_{\neg\text{cca}}, \mathcal{E}_{\neg\text{cca}}, \mathcal{E}_{\neg\text{cca}})$.** Here $\langle\cdot\rangle$ is an efficiently computable, injective encoding scheme.

The scheme $\Pi_{\neg\text{cca}}$ is by far no real-life AE construction (mainly due to its limited message space), its purpose is merely to act as a counter example. It can be verified, that for any adversary $\mathscr{A}$ with resources $(t, \mathbf{q_e}, \mathbf{q_d}, \boldsymbol{\sigma})$

$$\mathbf{Adv}^{\mathbf{vauth}(\tau_c)}_{\Pi_{\neg\text{cca}}}(\mathscr{A}) \leq \mathbf{Adv}^{\mathbf{prf}}_{F'}(t, q_e + q_d, \sigma) + \mathbf{q_d}[\tau_c]/2^{\tau_c}$$

for some $\mathscr{B}$ that runs in time $t + \gamma \cdot \sigma$ for some constant γ, makes no more than $q_e + q_d$ queries such that their data complexity is limited by σ. Here $q_e = \sum_{\tau \in \mathcal{I}_T} \boldsymbol{q_e}[\tau]$, $q_d = \sum_{\tau \in \mathcal{I}_T} \boldsymbol{q_d}[\tau]$ and $\sigma = \sum_{\tau \in \mathcal{I}_T} \boldsymbol{\sigma}[\tau]$. This is because every forgery attempt equals to guessing τ_c bits of an output of F', evaluated on a fresh input.

For confidentiality, we have that for any adversary $\mathscr{A}$ with resources $(t, \mathbf{q_e}, \boldsymbol{\sigma})$

$$\mathbf{Adv}^{\mathbf{vpriv}(\tau_c)}_{\Pi_{\neg\text{cca}}}(t, \mathbf{q_e}, \mathbf{q_d}, \boldsymbol{\sigma}) \leq \mathbf{Adv}^{\mathbf{prf}}_{F}(\mathscr{B}) + \mathbf{Adv}^{\mathbf{prf}}_{F'}(\mathscr{B}') + \mathbf{Adv}^{\widetilde{\mathbf{prp}}}_{\widetilde{E}}(\mathscr{C}) + 2q_e^2/2^n$$

for some $\mathscr{B}$ and $\mathscr{B}'$ that both run in time $t + \gamma \cdot \sigma$ for some constant γ, make no more than q_e queries such that their data complexity is limited by σ, and for some $\mathscr{C}$ that runs in time $t + \delta \cdot \sigma$ for some constant δ and makes no more than q_e queries, with the same q_e, q_d and σ as before.

The term $2q_e^2/2^n$ is composed of $q_e^2/2^n$ that comes from the RP-RF switch for the tweakable blockcipher and another $q_e^2/2^n$ that comes from extending the tweakspace to include stretch, using F (similar to Rogaway's XE construction [Rog04a]).

However, we can construct an adversary $\mathscr{A}_{\neg\text{cca}}$, that achieves IND-VCCA(τ_c) advantage close to 1. The strategy of $\mathscr{A}_{\neg\text{cca}}$ is as follows:

1. query $Z_1 \| T_1 \leftarrow \text{Enc}(N_1, A_1, \tau_c, M_1)$ with arbitrary N_1, A_1, M_1,
2. iterate through $T_1^* \in \{0,1\}^{\tau_{\min}}$ until $M_1^* \leftarrow \text{Dec}(N_1, A_1, \tau_{\min}, Z_1 \| T_1^*)$ returns $M_1^* \neq \perp$,
3. query $Z_2 \| T_2 \leftarrow \text{Enc}(N_2, A_2, \tau_c, M_2)$ with arbitrary N_2, A_2, M_2,
4. iterate through $T_2^* \in \{0,1\}^{\tau_{\min}}$ until $M_2^* \leftarrow \text{Dec}(N_2, A_2, \tau_{\min}, Z_2 \| T_2^*)$ returns $M_2^* \neq \perp$,
5. return 1 iff $M_1 \oplus M_1^* = M_2 \oplus M_2^*$ (otherwise return 0),

where $\tau_{\min} = \min(\mathcal{I}_T \backslash \{\tau_c\})$. $\mathscr{A}_{\neg\text{cca}}$ achieves $\mathbf{Adv}_{\Pi_{\neg\text{cca}}}^{\textbf{ind-vcca}(\tau_c)}(\mathscr{A}_{\neg\text{cca}}) = 1 - 2^{-n}$ and makes $2 \cdot 2^{\tau_{\min}}$ decryption and 2 encryption queries, all of fixed length.

As the amount of stretch τ has no effect on the encryption by $\widetilde{E}$, we can verify that

$$\begin{aligned} M_1 \oplus F(K_2, \langle \tau_c \rangle) &= M_1^* \oplus F(K_2, \langle \tau_{\min} \rangle) \\ M_2 \oplus F(K_2, \langle \tau_c \rangle) &= M_2^* \oplus F(K_2, \langle \tau_{\min} \rangle) \end{aligned}$$

The final condition in the **if**-statement verified by the adversary is always true for the real scheme. The probability of the same event in the "ideal" game is 2^{-n}. As a consequence of Theorem 8.4 and Proposition 8.6, we can state Corollary 8.7.[3] $\square$

8.5.3 Key-Equivalent Separation by Stretch

The notion of NVAE captures the immediate intuition about the security goal one expects to achieve using a nonce-based AE scheme with variable stretch. We now introduce a modular approach to *achieving* the notion.

Assume that an AE scheme is already known to be secure in the sense of the NAE model. What additional security property should such a scheme possess on top of NAE-security, so that it can achieve the full aim of being a NVAE-secure scheme? We formalize such a desirable property, naming it *key-equivalent separation by stretch* (KESS), which captures the intuition that for each value of stretch the scheme should behave as if keyed with a fresh secret key. The adversary must respect the relaxed nonce-requirement. The resources of interest for the KESS notion are $(t, \mathbf{q_e}, \mathbf{q_d}, \boldsymbol{\sigma})$, as defined for the NVAE(τ_c) notion in Section 8.5.1.

Definition 8.8 (KESS property). *Given a variable stretch AE scheme* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *with a stretch space* $\mathcal{I}_T$ *and an adversary* $\mathscr{A}$*, we define the advantage of* $\mathscr{A}$ *in breaking*

[3]The same attack strategy yields also $\mathbf{Adv}_{\Pi_{\neg\text{cca}}}^{\textbf{nvae}(\tau_c)}(\mathscr{A}_{\neg\text{cca}}) = 1 - 2^{-n}$.

```
proc initialize                        kess-R_Π
$K \leftarrow\$ \mathcal{K}$
$\mathcal{X} \leftarrow \varnothing$

oracle Enc$(N, A, \tau, M)$
if $(N, \tau) \in \mathcal{X}$ then
    return $\perp$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{(N, \tau)\}$
return $\mathcal{E}(K, N, A, \tau, M)$

oracle Dec$(N, A, \tau, C)$
return $\mathcal{D}(K, N, A, \tau, C)$
```

```
proc initialize                        kess-I_Π
for $\tau \in \mathcal{I}_T$ do
    $K_\tau \leftarrow\$ \mathcal{K}$

oracle Enc$(N, A, \tau, M)$
if $(N, \tau) \in \mathcal{X}$ then
    return $\perp$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{(N, \tau)\}$
return $\mathcal{E}(K_\tau, N, A, \tau, M)$

oracle Dec$(N, A, \tau, C)$
return $\mathcal{D}(K_\tau, N, A, \tau, C)$
```

Figure 8.8 – **Games defining KESS property of a variable-stretch AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$.** Note that the independent keying for each $\tau \in \mathcal{I}_T$ in game **kess-I**$_\Pi$ can be done by lazy sampling if needed.

the KESS property of Π *in a chosen ciphertext attack (with help of the games* **kess-*R*** *and* **kess-*I*** *in Figure 8.8) as*

$$\mathbf{Adv}_\Pi^{\mathbf{kess}}(\mathscr{A}) = \Pr\left[\mathscr{A}^{\mathbf{kess}\text{-}\boldsymbol{R}_\Pi} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathbf{kess}\text{-}\boldsymbol{I}_\Pi} \Rightarrow 1\right].$$

If $\mathbf{Adv}_\Pi^{\mathbf{kess}}(\mathscr{A}) \leq \epsilon$ *for all adversaries* $\mathscr{A}$ *whose resources are limited by* $(t, \boldsymbol{q}_e, \mathbf{q_d}, \boldsymbol{\sigma})$*, then we say that* Π *has the* $(\epsilon, t, \boldsymbol{q}_e, \mathbf{q_d}, \boldsymbol{\sigma})$*-KESS property.*

We note that KESS property on its own says nothing about AE security of a scheme. Indeed, a scheme whose encryption algorithm would simply output the message M followed by τ zeroes upon input (K, N, A, τ, M) would have perfect KESS property, but would be anything but AE secure in any sense. However, we show in Theorem 8.9 that when coupled with NAE security, KESS implies NVAE security. Informally, the **kess** notion takes care of interactions between queries with different values of stretch. Once this is done, we are free to argue that the queries with τ_c bits of stretch are "independent" of those with other values of stretch and will "inherit" the security level of $\Pi[\tau_c]$.

Theorem 8.9 (KESS $\wedge$ NAE $\Rightarrow$ NVAE). *Let* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *be a variable-stretch AE scheme and* $\mathscr{A}$ *and adversary with resources* $(t, \mathbf{q_e}, \mathbf{q_d}, \boldsymbol{\sigma})$*. We have that*

$$\mathbf{Adv}_\Pi^{\mathbf{nvae}(\tau_c)}(\mathscr{A}) \leq \mathbf{Adv}_\Pi^{\mathbf{kess}}(\mathscr{B}) + \mathbf{Adv}_{\Pi[\tau_c]}^{\mathbf{nae}}(\mathscr{C}),$$

for some $\mathscr{B}$ *with resources* $(t', \mathbf{q_e}, \mathbf{q_d}, \boldsymbol{\sigma})$ *and some* $\mathscr{C}$ *that runs in time* t''*, makes* $\boldsymbol{q}_e[\tau_c]$ *and* $\boldsymbol{q}_d[\tau_c]$ *encryption and decryption queries respectively with a total data complexity bounded by* $\boldsymbol{\sigma}[\tau_c]$*, such that* $t' = t + \beta \cdot q$ *and* $t'' = t + \gamma \cdot \sigma$ *for some constants* β *and* γ*,* $q = \sum_{\tau \in \mathcal{I}_T}(\boldsymbol{q}_e[\tau] + \mathbf{q_d}[\tau])$ *and* $\sigma = \sum_{\tau \in \mathcal{I}_T} \boldsymbol{\sigma}[\tau]$*.*

```
proc initialize
for $\tau \in \mathcal{I}_T$ do
    $K_\tau \leftarrow\$ \mathcal{K}$
$\mathcal{X} \leftarrow \varnothing$, $\mathcal{Y} \leftarrow \varnothing$

oracle Dec$(N, A, \tau, C)$
if $\tau = \tau_c$ and $(N, A, C) \in \mathcal{Y}$ then
    return $\bot$
return $\mathcal{D}(K_\tau, N, A, \tau, C)$

oracle Enc$(N, A, \tau, M)$
if $(N, \tau) \in \mathcal{X}$ then
        return $\bot$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{(N, \tau)\}$
$C \leftarrow \mathcal{E}(K_\tau, N, A, \tau, M)$
if $\tau = \tau_c$ then
        $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(N, A, C)\}$
return $C$
```

Figure 8.9 – **Security game nvae($\boldsymbol{\tau_c}$)-$\boldsymbol{G_\Pi}$.**

Proof. Consider the security game **nvae**(τ_c)-G defined in Figure 8.9. We have that

$$\mathbf{Adv}_{\Pi}^{\mathbf{nvae}(\tau_c)}(\mathscr{A}) = \Pr[\mathscr{A}^{\mathbf{nvae\text{-}R}(\tau_c)_\Pi} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{nvae}(\tau_c)\text{-}G_\Pi} \Rightarrow 1] + \Pr[\mathscr{A}^{\mathbf{nvae}(\tau_c)\text{-}G_\Pi} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{nvae\text{-}I}_\Pi(\tau_c)} \Rightarrow 1].$$

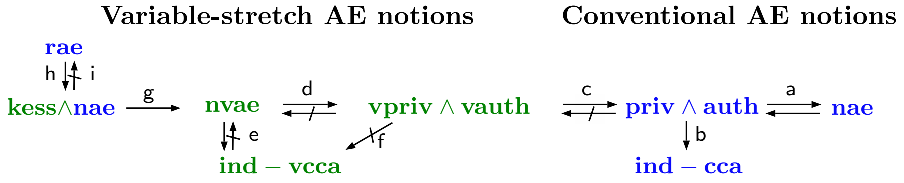
We first show that $\Pr[\mathscr{A}^{\mathbf{nvae\text{-}R}(\tau_c)_\Pi} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{nvae}(\tau_c)\text{-}G_\Pi} \Rightarrow 1] \leq \mathbf{Adv}_{\Pi}^{\mathbf{kess}}(\mathscr{B})$. The NVAE adversary $\mathscr{A}$ can be straightforwardly reduced to $\mathscr{B}$. Any query of $\mathscr{A}$ is directly answered with $\mathscr{B}$'s own oracles, except for decryption queries with expansion of τ_c bits, whose output is trivially known from previous encryption queries; here $\mathscr{B}$ returns $\bot$ to $\mathscr{A}$. At the end, $\mathscr{B}$ outputs whatever $\mathscr{A}$ outputs. If $\mathscr{B}$ interacts with **kess-R**$_\Pi$ then it perfectly simulates **nvae-R**$(\tau_c)_\Pi$ for $\mathscr{A}$. If $\mathscr{B}$ interacts with **kess-I**$_\Pi$ then it perfectly simulates **nvae**(τ_c)-G_Π.

We next show that $\Pr[\mathscr{A}^{\mathbf{nvae}(\tau_c)\text{-}G_\Pi} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{nvae\text{-}I}_\Pi(\tau_c)} \Rightarrow 1] \leq \mathbf{Adv}_{\Pi[\tau_c]}^{\mathbf{nae}}(\mathscr{C})$. The adversary $\mathscr{A}$ can be reduced to $\mathscr{C}$ in the following way. When $\mathscr{A}$ issues a query with expansion τ_c, $\mathscr{C}$ answers it with its own oracles. For other amounts of stretch $\tau \neq \tau_c$, $\mathscr{C}$ first checks if there were previous queries with τ bits of stretch. If not, it samples a fresh key K_τ. $\mathscr{C}$ then processes the query with the real (encryption or decryption) algorithm of Π and the key K_τ, making sure that encryption queries comply with the relaxed nonce requirement. If $\mathscr{C}$ interacts with **nae-R**$_{\Pi[\tau_c]}$ then it perfectly simulates **nvae**(τ_c)-G_Π for $\mathscr{A}$. If $\mathscr{C}$ interacts with **nae-I**$_{\Pi[\tau_c]}$ then it perfectly simulates **nvae-I**$(\tau_c)_\Pi$. This yields the desired result. □

Remark 11. *An RAE secure scheme* Π *will always have the* **kess** *property (see Appendix B.4 for definition of RAE security). To see why, note that replacing* Π *by a collection of random injections in both the* **kess-*R***$_\Pi$ *and* **kess-*I***$_\Pi$ *games will not increase the advantage significantly, as that would contradict* Π*'s RAE security. After the replacement, the two games will be indistinguishable. On the other hand,* **kess** *property does not guarantee RAE security; the scheme* OCBv *described in Section 8.6 can serve as a counter-example, because it does not tolerate nonce reuse.*

8.5.4 Relations among Notions

We address the relations between the notions that are newly defined in Chapter 8 and the previously existing notions throughout the Section 8.5. The summary of established relations can be found in Figure 8.10.



$$\begin{tikzcd}
\textbf{rae} \arrow[d, shift right, "h"'] & & & & \\
\textbf{kess}\wedge\textbf{nae} \arrow[u, shift right, "/" marking, "i"'] \arrow[r, "g"] & \textbf{nvae} \arrow[r, shift left, "d"] \arrow[d, shift right] & \textbf{vpriv} \wedge \textbf{vauth} \arrow[l, shift left, "/" marking] \arrow[dl, "/" marking, "f"] \arrow[r, shift left, "c"] & \textbf{priv} \wedge \textbf{auth} \arrow[l, shift left, "/" marking] \arrow[r, shift left, "a"] \arrow[d, "b"] & \textbf{nae} \arrow[l, shift left] \\
& \textbf{ind} - \textbf{vcca} \arrow[u, shift right, "/" marking, "e"'] & & \textbf{ind} - \textbf{cca} &
\end{tikzcd}$$

Figure 8.10 – Relations among notions for nonce-based AE with and without variable stretch. Previous works: a[RS06b], b[BN00]. This paper: c (Remark 10, attacks in Section 8.4), d (Remark 10, Corollary 8.7), e (Theorem 8.4, Remark 9), f (Proposition 8.6), g (Theorem 8.9), h, i (Remark 11 together with[HKR15]).

8.5.5 A Short Guide to NVAE

Interpretation of the NVAE(τ_c) security advantage. The games defining the notion of NVAE are parameterized by a constant, but arbitrary amount of stretch τ_c from the stretch space $\mathcal{I}_T$ of the AE scheme Π in question. In the **nvae**(τ_c)-$\mathbf{I}_\Pi$ security game, only queries expanded by τ_c bits will be subjected to "idealization". For all other expansions, we give the adversary complete freedom to ask any queries it wants (except for those breaking the nonce-requirement), but their behaviour is the same in both security games. An NVAE security bound that is expressed as a function of τ_c and *assumes no particular value or constraint for* τ_c will therefore tell us, what security guarantees can we expect from queries stretched by τ_c bits specifically, for any $\tau_c \in \mathcal{I}_T$.

Looking at the security bound itself, we are able to tell if there are any undesirable interactions between queries with different amounts of stretch. This is best illustrated by revisiting the problems and forgery attack from Section 8.4 in the NVAE security model.

Attacks in NAE model. With the formal framework defined, we revisit the attacks from Section 8.4 and analyse the advantage they achieve, as well as the resources they require. Consider the original, unmodified scheme OCB [KR11], that produces the tag by truncating an n-bit (with $n > \tau$) to τ bits. In case of simultaneous use of two (or more) amounts of stretch $\tau_1 < \tau_2$ with the same key, we can forge a ciphertext stretched by τ_1 bits by truncating an existing ciphertext stretched by τ_2 bits. This would correspond to an attack with an **nvae**(τ_1) advantage of 1 and constant resources.

If the same scheme is treated with the heuristic measures, e.g., nonce-stealing or encoding τ in AD, from Section 8.4.1 (let's call it hOCB), we consider the forgery attack from Section 8.4.2. Assume that there are four instances of hOCB, with $32, 64, 96$ and

128 bit tags. To make a forgery with 128-bit tag, we have to forge a ciphertext with 32 bits of expansion and then exhaustively search for three 32-bit extensions of this forgery. This gives us an **nvae**(128) advantage equal to 1, requiring 4 encryption queries, $3 \cdot 2^{32}$ verification queries with stretch other than 128 bits and 2^{32} decryption queries stretched by 128 bits. The effort necessary for such a forgery is clearly smaller than we could hope for, especially in the amount of verification queries stretched by the challenge amount of bits (i.e. 128).

"Good" bounds. After seeing examples of attacks, one may wonder: what kind of NVAE security bound should we expect from a secure variable-stretch scheme? For every scheme, it must be always possible to guess a ciphertext with probability $2^{-\tau_c}$. Thus the bound must always contain a term of the form $c \cdot (\boldsymbol{q}_d[\tau_c])^\alpha / 2^{\tau_c}$ for some positive constants c and α, or some more complicated function of this.

Even though the security level for τ_c-stretched queries should be independent of any other queries, it is usually unavoidable to have a gradual increase of advantage with every query made by the adversary. This increase can generally depend on all of the adversarial resources, but should not depend on τ_c itself.

An example of a secure scheme's NVAE(τ_c) bound can be found in Theorem 8.15. It consist of the fraction $(\boldsymbol{q}_d[\tau_c] \cdot 2^{n-\tau_c})/(2^n - 1) \approx \boldsymbol{q}_d[\tau_c]/2^{\tau_c}$, advantage bounds for the used blockcipher and a birthday-type term that grows with the total amount of data processed. We see, that queries stretched by $\tau \neq \tau_c$ bits will not unexpectedly increase adversary's chances to break OCBv, and that the best attack strategy to forge with τ_c bits of stretch is simple tag-guessing.

8.6 Achieving AE with Variable Stretch

We demonstrate that the security of AE schemes in the sense of the NVAE notion is easily achievable by introducing a practical and secure scheme. Rather than constructing a scheme from the scratch, we modify an existing, well-established scheme and follow a modular approach to analyse its security in presence of variable stretch. The modification we propose is general enough to be applicable to most of the AE schemes based on a tweakable primitive (e.g. a tweakable blockcipher).

OCB mode for tweakable blockcipher The Offset Codebook mode of operation for a tweakable blockcipher (ΘCB) is a nonce-based AE scheme proposed by Krovetz and Rogaway [KR11] (there are subtle differences from the prior versions of OCB [RBBK01, Rog04a]). It is parameterized by a tweakable blockcipher $\widetilde{E} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$ (we will denote the inverse of $\widetilde{E}$ as $\widetilde{E}^{-1} = \widetilde{D}$) and a tag length $0 \leq \tau \leq n$. The tweak space of $\widetilde{E}$ is of the form $\mathcal{T} = \mathcal{N} \times \mathbb{N}_0 \times \{0,1,2,3\} \cup \mathbb{N}_0 \times \{0,1,2,3\}$ for a finite set $\mathcal{N}$. The encryption and the decryption algorithms of ΘCB$[\widetilde{E}, \tau]$ are described in Figure 8.11. The AE security of ΘCB is formally stated in Lemma 8.10.

```
procedure $\mathcal{E}_K(N, A, M)$
    if $N \notin \mathcal{N}$ then
        return $\perp$
    end if
    $M_1 \| M_2 \cdots M_m \| M_* \xleftarrow{n} M$
    Sum $\leftarrow 0^n$, $C_* \leftarrow \varepsilon$
    for $i \leftarrow 1$ to $m$ do
        $C_i \leftarrow \widetilde{E}_K^{N,i,0}(M_i)$
        Sum $\leftarrow$ Sum $\oplus M_i$
    end for
    if $M_* = \varepsilon$ then
        Final $\leftarrow \widetilde{E}_K^{N,m,2}$(Sum)
    else
        Pad $\leftarrow \widetilde{E}_K^{N,m,1}(0^n)$
        $C_* \leftarrow M_* \oplus \mathsf{left}_{|M_*|}$ (Pad)
        Sum $\leftarrow$ Sum $\oplus M_* \| 10^*$
        Final $\leftarrow \widetilde{\mathrm{E}}_{\mathrm{K}}^{\mathrm{N,m,3}}$(Sum)
    end if
    Auth $\leftarrow$ HASH$_K(A)$
    $T \leftarrow \mathsf{left}_\tau$ (Final $\oplus$ Auth)
    return $C_1 \| C_2 \| \cdots \| C_m \| C_* \| T$
end procedure

procedure HASH$_K(A)$
    Sum $\leftarrow 0^n$
    $A_1 \| A_2 \cdots A_m \| A_* \xleftarrow{n} A$
    for $i \leftarrow 1$ to $m$ do
        Sum $\leftarrow$ Sum $\oplus \widetilde{E}_K^{i,0}(A_i)$
    end for
    if $A_* \neq \varepsilon$ then
        Sum $\leftarrow$ Sum $\oplus \widetilde{E}_K^{m,1}(A_* \| 10^*)$
    end if
    return Sum
end procedure

procedure $\mathcal{D}_K(N, A, \mathsf{C})$
    if $N \notin \mathcal{N}$ or $|\mathsf{C}| < \tau$ then
        return $\perp$
    end if
    $C \leftarrow \mathsf{left}_{|\mathsf{C}|-\tau}(\mathsf{C})$ ; $T \leftarrow \mathsf{left}_\tau(\mathsf{C})$
    $C_1 \| C_2 \cdots C_m \| C_* \xleftarrow{n} C$
    Sum $\leftarrow 0^n$, $M_* \leftarrow \varepsilon$
    for $i \leftarrow 1$ to $m$ do
        $M_i \leftarrow \widetilde{D}_K^{N,\tau,i,0}(C_i)$
        Sum $\leftarrow$ Sum $\oplus M_i$
    end for
    if $C_* = \varepsilon$ then
        Final $\leftarrow \widetilde{E}_K^{N,m,2}$(Sum)
    else
        Pad $\leftarrow \widetilde{E}_K^{N,m,1}(0^n)$
        $M_* \leftarrow C_* \oplus \mathsf{left}_{|C_*|}$ (Pad)
        Sum $\leftarrow$ Sum $\oplus M_* \| 10^*$
        Final $\leftarrow \widetilde{\mathrm{E}}_{\mathrm{K}}^{\mathrm{N,m,3}}$(Sum)
    end if
    Auth $\leftarrow$ HASH$_K(A)$
    $T' \leftarrow \mathsf{left}_\tau$ (Final $\oplus$ Auth)
    if $T = T'$ then
        return $C_1 \| \cdots \| C_m \| C_* \| T$
    else
        return $\perp$
    end if
end procedure
```

Figure 8.11 – **Definition of $\boldsymbol{\Theta}$CB[$\widetilde{\boldsymbol{E}}, \boldsymbol{\tau}$].**

Lemma 8.10 (Lemma 2, [KR11]). *Let $\widetilde{E} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$ be a tweakable blockcipher with $\mathcal{T} = \mathcal{N} \times \mathbb{N}_0 \times \{0,1,2,3\} \cup \mathbb{N}_0 \times \{0,1,2,3\}$. Let $\tau \in \{0,\ldots,n\}$. Let $\mathscr{A}$ be an adversary that runs in time t and makes q_e encryption queries of no more than σ bits. Let further $\mathscr{A}'$ be an adversary that runs in time t, makes q_e encryption queries and q_d decryption queries of no more than σ bits in total. Then we have that*

$$\mathbf{Adv}^{\mathbf{priv}}_{\Theta\mathrm{CB}[\widetilde{E},\tau]}(\mathscr{A}) \leq \mathbf{Adv}^{\widetilde{\mathbf{sprp}}}_{\widetilde{E}}(\mathscr{B}),$$

$$\mathbf{Adv}^{\mathbf{auth}}_{\Theta\mathrm{CB}[\widetilde{E},\tau]}(\mathscr{A}') \leq \mathbf{Adv}^{\widetilde{\mathbf{sprp}}}_{\widetilde{E}}(\mathscr{B}') + q_d \cdot \frac{2^{n-\tau}}{2^n - 1},$$

for some $\mathscr{A}$ that runs in time $t + \gamma \cdot \sigma$ for a constant γ and makes at most q_p queries, and some $\mathscr{A}'$ that runs in time $t + \gamma' \cdot \sigma$ for a constant γ' and makes at most q_a queries where $q_p \leq \lceil \sigma/n \rceil + 2 \cdot q_e$, and $q_a \leq \lceil \sigma/n \rceil + 2 \cdot (q_e + q_d)$.

Thanks to Lemma 2.9, we state as a corollary of Lemma 8.10 that $\mathbf{Adv}^{\mathbf{nae}}_{\Theta\mathrm{CB}[\widetilde{E},\tau]}(\mathscr{A}) \leq \mathbf{Adv}^{\widetilde{\mathbf{sprp}}}_{\widetilde{E}}(\mathscr{B}) + q_d \frac{2^{n-\tau}}{2^n-1}$ for an $\mathscr{A}$ with resources as in Lemma 8.10, and a $\mathscr{B}$ that runs in time $t + \gamma \cdot \sigma$ for a constant γ and makes at most $\lceil \sigma/n \rceil + 2 \cdot (q_e + q_d)$ queries.

OCB mode with variable-stretch security We introduce ΘCBv (variable-stretch-ΘCB), a variable-stretch AE scheme obtained by slightly modifying ΘCB.

The tweakable blockcipher mode of operation ΘCBv is parameterized by a tweakable blockcipher $\widetilde{E} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$ and a stretch-space $\mathcal{I}_T \subseteq \{0,1,\dots,n\}$. The current tweak space $\mathcal{T}$ is different than the one for ΘCB; it is of the form $\mathcal{T} = \mathcal{N} \times \mathcal{I}_T \times \mathbb{N}_0 \times \{0,1,2,3\} \cup \mathcal{I}_T \times \mathbb{N}_0 \times \{0,1,2,3\}$. The encryption and decryption algorithms of ΘCBv are exactly the same as those of ΘCB, except that they now allow selectable stretch and that every call to $\widetilde{E}$ is now tweaked by τ, in addition to the other tweak components. Both algorithms are described in Figure 8.12. An illustration of the encryption algorithm is depicted in Figure 8.13.

Thanks to Theorem 8.9, establishing the NVAE security of ΘCBv requires little effort. The corresponding result is stated in Theorem 8.11.

Theorem 8.11. *Let $\widetilde{E} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$ be a tweakable blockcipher with $\mathcal{T} = \mathcal{N} \times \mathcal{I}_T \times \mathbb{N}_0 \times \{0,1,2,3\} \cup \mathcal{I}_T \times \mathbb{N}_0 \times \{0,1,2,3\}$. Let $\mathscr{A}$ be an NVAE-adversary with resources bounded by $(t, \mathbf{q_e}, \mathbf{q_d}, \boldsymbol{\sigma})$. Then we have that*

$$\mathbf{Adv}^{\mathbf{nvae}(\tau_c)}_{\Theta\mathrm{CBv}[\widetilde{E}]}(\mathscr{A}) \leq \mathbf{Adv}^{\widetilde{\mathbf{sprp}}}_{\widetilde{E}}(\mathscr{B}) + \sum_{\tau \in \mathcal{I}_T} \mathbf{Adv}^{\widetilde{\mathbf{sprp}}}_{\widetilde{E}}(\mathscr{C}_\tau) + \mathbf{Adv}^{\widetilde{\mathbf{sprp}}}_{\widetilde{E}}(\mathscr{C}_{\tau_c}) + \boldsymbol{q}_d[\tau_c] \cdot \frac{2^{n-\tau_c}}{2^n - 1}.$$

for some $\mathscr{B}$ that runs in time $t + \beta \cdot q$ and makes q queries, and some $\mathscr{C}_\tau$ that runs in time $t + \gamma_\tau \cdot \boldsymbol{q}[\tau]$ and makes $\boldsymbol{q}[\tau]$ queries for $\tau \in \mathcal{I}_T$, where $\boldsymbol{q}[\tau] = \lceil \boldsymbol{\sigma}[\tau]/n \rceil + 2 \cdot (\boldsymbol{q}_e[\tau] + \boldsymbol{q}_d[\tau])$ for $\tau \in \mathcal{I}_T$, and $q = \sum_{\tau \in \mathcal{I}_T} \boldsymbol{q}[\tau]$, and β and γ_τ for $\tau \in \mathcal{I}_T$ are constants.

Proof. We observe that if we fix the expansion value to τ_c in all queries, the nonce-based AE scheme $(\Theta\mathrm{CBv}[\widetilde{E}])[\tau_c]$ that we get will be identical with the scheme $\Theta\mathrm{CB}[\widetilde{E}, \tau_c]$. The result follows from this observation and the results of Lemmas 8.10 and 8.12, and Theorem 8.9. □

Lemma 8.12. *Let $\widetilde{E} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$ be a tweakable blockcipher with $\mathcal{T} = \mathcal{N} \times \mathcal{I}_T \times \mathbb{N}_0 \times \{0,1,2,3\} \cup \mathcal{I}_T \times \mathbb{N}_0 \times \{0,1,2,3\}$. Let $\mathscr{A}$ be a KESS-adversary with resources bounded by $(t, \mathbf{q_e}, \mathbf{q_d}, \boldsymbol{\sigma})$. Then we have that*

$$\mathbf{Adv}^{\mathbf{nvae}(\tau_c)}_{\Theta\mathrm{CBv}[\widetilde{E}]}(\mathscr{A}) \leq \mathbf{Adv}^{\widetilde{\mathbf{sprp}}}_{\widetilde{E}}(\mathscr{B}) + \sum_{\tau \in \mathcal{I}_T} \mathbf{Adv}^{\widetilde{\mathbf{sprp}}}_{\widetilde{E}}(\mathscr{C}_\tau)$$

for some $\mathscr{B}$ that runs in time $t + \beta \cdot q$ and makes q queries, and some $\mathscr{C}_\tau$ that runs in time $t + \gamma_\tau \cdot \boldsymbol{q}[\tau]$ and makes $\boldsymbol{q}[\tau]$ queries for $\tau \in \mathcal{I}_T$, where $\boldsymbol{q}[\tau] = \lceil \boldsymbol{\sigma}[\tau]/n \rceil + 2 \cdot (\boldsymbol{q}_e[\tau] + \boldsymbol{q}_d[\tau])$ for $\tau \in \mathcal{I}_T$, and $q = \sum_{\tau \in \mathcal{I}_T} \boldsymbol{q}[\tau]$, and β and γ_τ for $\tau \in \mathcal{I}_T$ are constants.

```
procedure $\mathcal{E}_K(N, A, \tau, M)$
    if $N \notin \mathcal{N}$ then
        return $\perp$
    end if
    $M_1\|M_2\cdots M_m\|M_* \xleftarrow{n} M$
    $\mathrm{Sum} \leftarrow 0^n$, $C_* \leftarrow \varepsilon$
    for $i \leftarrow 1$ to $m$ do
        $C_i \leftarrow \widetilde{E}_K^{N,\tau,i,0}(M_i)$
        $\mathrm{Sum} \leftarrow \mathrm{Sum} \oplus M_i$
    end for
    if $M_* = \varepsilon$ then
        $\mathrm{Final} \leftarrow \widetilde{E}_K^{N,\tau,m,2}(\mathrm{Sum})$
    else
        $\mathrm{Pad} \leftarrow \widetilde{E}_K^{N,\tau,m,1}(0^n)$
        $C_* \leftarrow M_* \oplus \mathsf{left}_{|M_*|}(\mathrm{Pad})$
        $\mathrm{Sum} \leftarrow \mathrm{Sum} \oplus M_*\|10^*$
        $\mathrm{Final} \leftarrow \widetilde{\mathrm{E}}_{\mathrm{K}}^{\mathrm{N},\tau,\mathrm{m},3}(\mathrm{Sum})$
    end if
    $\mathrm{Auth} \leftarrow \mathrm{HASH}_K(A)$
    $T \leftarrow \mathsf{left}_\tau(\mathrm{Final} \oplus \mathrm{Auth})$
    return $C_1\|C_2\|\cdots\|C_m\|C_*\|T$
end procedure

procedure $\mathrm{HASH}_K(A, \tau)$
    $\mathrm{Sum} \leftarrow 0^n$
    $A_1\|A_2\cdots A_m\|A_* \xleftarrow{n} A$
    for $i \leftarrow 1$ to $m$ do
        $\mathrm{Sum} \leftarrow \mathrm{Sum} \oplus \widetilde{E}_K^{\tau,i,0}(A_i)$
    end for
    if $A_* \neq \varepsilon$ then
        $\mathrm{Sum} \leftarrow \mathrm{Sum} \oplus \widetilde{E}_K^{\tau,m,1}(A_*\|10^*)$
    end if
    return Sum
end procedure

procedure $\mathcal{D}_K(N, A, \tau, \mathsf{C})$
    if $N \notin \mathcal{N}$ or $|\mathsf{C}| < \tau$ then
        return $\perp$
    end if
    $C \leftarrow \mathsf{left}_{|\mathsf{C}|-\tau}(\mathsf{C})$ ; $T \leftarrow \mathsf{left}_\tau(\mathsf{C})$
    $C_1\|C_2\cdots C_m\|C_* \xleftarrow{n} C$
    $\mathrm{Sum} \leftarrow 0^n$, $M_* \leftarrow \varepsilon$
    for $i \leftarrow 1$ to $m$ do
        $M_i \leftarrow \widetilde{D}_K^{N,\tau,i,0}(C_i)$
        $\mathrm{Sum} \leftarrow \mathrm{Sum} \oplus M_i$
    end for
    if $C_* = \varepsilon$ then
        $\mathrm{Final} \leftarrow \widetilde{E}_K^{N,\tau,m,2}(\mathrm{Sum})$
    else
        $\mathrm{Pad} \leftarrow \widetilde{E}_K^{N,\tau,m,1}(0^n)$
        $M_* \leftarrow C_* \oplus \mathsf{left}_{|C_*|}(\mathrm{Pad})$
        $\mathrm{Sum} \leftarrow \mathrm{Sum} \oplus M_*\|10^*$
        $\mathrm{Final} \leftarrow \widetilde{\mathrm{E}}_{\mathrm{K}}^{\mathrm{N},\tau,\mathrm{m},3}(\mathrm{Sum})$
    end if
    $\mathrm{Auth} \leftarrow \mathrm{HASH}_K(A)$
    $T' \leftarrow \mathsf{left}_\tau(\mathrm{Final} \oplus \mathrm{Auth})$
    if $T = T'$ then
        return $C_1\|\cdots\|C_m\|C_*\|T$
    else
        return $\perp$
    end if
end procedure
```

Figure 8.12 – **Definition of $\Theta\mathbf{CBv}[\widetilde{E}]$.** Changes from ΘCB highlighted in red.

Proof. Let $\mathscr{A}$ be a KESS adversary with indicated resources. We define the games $\widetilde{\textbf{kess-R}}$ and $\widetilde{\textbf{kess-I}}$ by replacing the tweakable blockcipher $\widetilde{E}$ by an ideal primitive in **kess-R** and the **kess-I** respectively; i.e. we sample an independent random tweakable permutation $\widetilde{\pi}_K \leftarrow\$ \widetilde{\mathrm{Perm}}(\mathcal{T}, 2^n)$ for every $K \in \mathcal{K}$ and use $\widetilde{\pi}_K$ instead of $\widetilde{E}_K$ for each K.

We have that

$$\begin{aligned}\mathbf{Adv}^{\mathbf{nvae}(\tau_c)}_{\Theta\mathrm{CBv}[\widetilde{E}]}(\mathscr{A}) = \Pr\left[\mathscr{A}^{\textbf{kess-R}} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\widetilde{\textbf{kess-R}}} \Rightarrow 1\right]\\ + \Pr\left[\mathscr{A}^{\widetilde{\textbf{kess-R}}} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\widetilde{\textbf{kess-I}}} \Rightarrow 1\right]\\ + \Pr\left[\mathscr{A}^{\widetilde{\textbf{kess-I}}} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\textbf{kess-I}} \Rightarrow 1\right].\end{aligned}$$

The gap $\Pr\left[\mathscr{A}^{\mathbf{kess\text{-}R}} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\widetilde{\mathbf{kess\text{-}R}}} \Rightarrow 1\right]$ is bounded by $\mathbf{Adv}^{\widetilde{\mathbf{sprp}}}_{\widetilde{E}}(\mathscr{B})$ by a standard reduction.

To bound the gap $\Pr\left[\mathscr{A}^{\widetilde{\mathbf{kess\text{-}I}}} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathbf{kess\text{-}I}} \Rightarrow 1\right]$, we observe that the replacement can be done gradually, for one value of stretch at a time. We define games $\mathbf{kess\text{-}I}_i$ for $i = 0, \ldots, |\mathcal{I}_T|$. In the game $\mathbf{kess\text{-}I}_i$, we replace the calls to $\widetilde{E}_{K_{\tau_j}}$ by calls to $\widetilde{\pi}_{K_{\tau_j}}$ for $j = 1, \ldots, i$ (using some fixed ordering of elements of $\mathcal{I}_T$). Thus $\mathbf{kess\text{-}I}_0 = \mathbf{kess\text{-}I}$ and $\mathbf{kess\text{-}I}_{|\mathcal{I}_T|} = \widetilde{\mathbf{kess\text{-}I}}$. For each $i = 1, \ldots, |\mathcal{I}_T|$, the gap $\Pr\left[\mathscr{A}^{\mathbf{kess\text{-}I}_i} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathbf{kess\text{-}I}_i} \Rightarrow 1\right]$ is bounded by $\mathbf{Adv}^{\widetilde{\mathbf{sprp}}}_{\widetilde{E}}(\mathscr{C}_{\tau_i})$. Thus, by a standard hybrid argument, the cumulative gap will be bounded by $\sum_{\tau \in \mathcal{I}_T} \mathbf{Adv}^{\widetilde{\mathbf{sprp}}}_{\widetilde{E}}(\mathscr{C}_\tau)$.

Once $\widetilde{E}$ is replaced by a collection of random tweakable permutations in both games, we observe that both $\widetilde{\mathbf{kess\text{-}R}}$ and $\widetilde{\mathbf{kess\text{-}I}}$ produce identical distributions. This is because both in $\widetilde{\mathbf{kess\text{-}R}}$ and in $\widetilde{\mathbf{kess\text{-}I}}$, any two queries with any two unequal amounts of stretch τ_1 and τ_2 will be processed by two independent collections of random permutations (thanks to the separation of queries with different amounts of stretch by tweaks). □

Instantiating $\widetilde{E}$. In order to obtain a real-world scheme, we need to instantiate the tweakable blockcipher $\widetilde{E}$. The scheme OCB uses the XEX construction [Rog04a] that turns an ordinary blockcipher $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ into a tweakable blockcipher $\widetilde{E} = \text{XEX}[E]$ with $\widetilde{E} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$. A call to $\widetilde{E} = \text{XEX}[E]$ is evaluated in two ways, depending on the tweak:

$$\widetilde{E}_K^{N,i,j}(X) = E_K(X \oplus \Delta_{N,i,j}) \oplus \Delta_{N,i,j}, \quad \text{or} \quad \widetilde{E}_K^{i,j}(X) = E_K(X \oplus \Delta_{i,j}).$$

In each call, the input (and in some cases also the output) of the blockcipher E is masked with special Δ-values, derived from the tweak and the secret key. A function $H : \mathcal{K} \times \{0,1\}^{<n} \to \{0,1\}^n$ defined as $H(K,N) = E_K(N\|10^*)$ is used in the computation of the masking values.[4] In the following, all multiplications are done in $\mathsf{GF}(2^n)$ with some fixed representation. The masking Δ-values of the original OCB are computed as follows:

$$\begin{aligned}
&\Delta_{N,0,0} = H(K,N),\\
&\Delta_{N,i+1,0} = \Delta_{N,i,0} \oplus L[\mathtt{ntz}(i+1)] \text{ for } i \geq 0,\\
&\Delta_{N,i,j} = \Delta_{N,i,0} \oplus j \cdot L_* \text{ for } j \in \{0,1,2,3\},\\
&\Delta_{0,0} = 0^n,\\
&\Delta_{i+1,0} = \Delta_{i,0} \oplus L[\mathtt{ntz}(i+1)] \text{ for } i \geq 0,\\
&\Delta_{i,j} = \Delta_{i,0} \oplus j \cdot L_* \text{ for } j \in \{0,1,2,3\},
\end{aligned}$$

where $L_* = E_K(0^n)$, $L[0] = 2^2 \cdot L_*$, $L[\ell] = 2 \cdot L[\ell - 1]$ for $\ell > 0$ and $\mathtt{ntz}(i)$ denotes the number of trailing zeros function (see Section 2.1 for definition). The security of the XEX construction is stated in Lemma 8.13.

[4]A function is used in the latest version of OCB [KR11], we opted for $E_K(\cdot)$ for the sake of simplicity.

Lemma 8.13 (Lemma 3, [KR11])**.** *Let $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ be a blockcipher and $\mathcal{T} = \mathcal{N} \times \mathbb{N}_0 \times \{0,1,2,3\} \cup \mathbb{N}_0 \times \{0,1,2,3\}$. Let $\mathscr{A}$ be an adversary that runs in time at most t, asks at most q queries, never asks queries with i-component exceeding 2^{n-5} and never asks decryption queries with tweaks from $\mathbb{N}_0 \times \{0,1,2,3\}$. Then*

$$\mathbf{Adv}^{\widetilde{\mathbf{sprp}}}_{\mathrm{XEX}[E]}(\mathscr{A}) \leq \mathbf{Adv}^{\pm\mathrm{prp}}_{E}(\mathscr{B}) + \frac{6q^2}{2^n}$$

for an adversary $\mathscr{B}$ that makes at most $2q$ queries and runs in time bounded by $t + \gamma \cdot q$ with some constant γ.

Extending the tweaks with $\boldsymbol{\tau}$. In order to instantiate ΘCBv, we need to extend the tweaks of $\widetilde{E}$ with a fourth component: τ. To this end, we propose XEX′, which is obtained by a slight modification of the XEX construction. Informally, we expand the domain of the "j-part" of tweaks and represent it as $\mathcal{I}_T \times \{0,1,2,3\}$, compensating for this by decreasing the maximal value of i.

The tweakable blockcipher $\widetilde{E}' = \mathrm{XEX}'[E]$ is defined as follows. We again use the function $H(K,N)$. We uniquely label each element of $\mathcal{I}_T$ by an integer with a bijection $\lambda : \mathcal{I}_T \to \{0,1,\dots,|\mathcal{I}_T|-1\}$. We define $m = \lceil \log_2 |\mathcal{I}_T| \rceil$, $L_* = E_K(0^n)$, $L_\tau = \lambda(\tau) \cdot 2^2 \cdot L_*$ for $\tau \in \mathcal{I}_T$, $L[0] = 2^{2+m} \cdot L_*$, and $L[\ell] = 2 \cdot L[\ell-1]$ for $\ell > 0$. The masking Δ-values are computed as follows:

$$\begin{aligned}
&\Delta_{N,0,0,0} = H(K,N),\\
&\Delta_{N,\tau,0,0} = \Delta_{N,0,0,0} \oplus L_\tau,\\
&\Delta_{N,\tau,i+1,0} = \Delta_{N,\tau,i,0} \oplus L[\mathtt{ntz}(i+1)] \text{ for } i \geq 0,\\
&\Delta_{N,\tau,i,j} = \Delta_{N,\tau,i,0} \oplus j \cdot L_* \text{ for } j \in \{0,1,2,3\},\\
&\Delta_{\tau,0,0} = L_\tau,\\
&\Delta_{\tau,i+1,0} = \Delta_{\tau,i,0} \oplus L[\mathtt{ntz}(i+1)] \text{ for } i \geq 0,\\
&\Delta_{\tau,i,j} = \Delta_{\tau,i,0} \oplus j \cdot L_* \text{ for } j \in \{0,1,2,3\}.
\end{aligned}$$

A call to $\widetilde{E}'$ is evaluated as follows:

$$\widetilde{E}'^{N,\tau,i,j}_K(X) = E_K(X \oplus \Delta_{N,\tau,i,j}) \oplus \Delta_{N,\tau,i,j}, \quad \text{or} \quad \widetilde{E}'^{\tau,i,j}_K(X) = \quad E_K(X \oplus \Delta_{\tau,i,j}).$$

The security result for XEX′ construction is stated in Lemma 8.14.

Lemma 8.14. *Let $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ be a blockcipher and $\mathcal{T} = \mathcal{N} \times \mathcal{I}_T \times \mathbb{N}_0 \times \{0,1,2,3\} \cup \mathcal{I}_T \times \mathbb{N}_0 \times \{0,1,2,3\}$ for some finite, non-empty $\mathcal{I}_T \subseteq \mathbb{N}_0$. Let $\mathscr{A}$ be an adversary that runs in time at most t, asks at most q queries, never asks queries with i-component exceeding $2^{n-(5+\lceil \log_2 |\mathcal{I}_T| \rceil)}$ and never asks decryption queries with tweaks from $\mathcal{I}_T \times N_0 \times \{0,1,2,3\}$. Then*

$$\mathbf{Adv}^{\widetilde{\mathbf{sprp}}}_{\mathrm{XEX}'[E]}(\mathscr{A}) \leq \mathbf{Adv}^{\pm\mathrm{prp}}_{E}(\mathscr{B}) + \frac{6q^2}{2^n}$$

for an adversary $\mathscr{B}$ that makes at most $2q$ queries and runs in time bounded by $t + \gamma \cdot q$ for some constant γ.

The treatment of the τ-tweak component in XEX$'$ construction is equivalent to a one where we would injectively encode τ, j into a single integer $j' = 2^2\tau + j < 2^{2+m}$. This is the same approach that was used to extend the tweak space of XE construction for MR-OMD and p-OMD in Sections 4.5 ans 5.5, where it is shown that the essential properties of the masking values necessary for the security proof of [KR11] are preserved. The same arguments apply here, so we omit the proof of Lemma 8.14.

OCBv: practical AE with variable stretch. We define the blockcipher mode of operation OCBv, a variable-stretch AE scheme. OCBv is parameterized by a blockcipher E and a stretch space $\mathcal{I}_T$. It is obtained by instantiating the tweakable blockcipher in ΘCBv by the XEX$'$ construction, i.e. OCBv$[E]$ = ΘCBv[XEX$'[E]$] and the claim on its security is stated in Theorem 8.15.

Theorem 8.15. *Let $\widetilde{E} : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ be a blockcipher. Let $\mathscr{A}$ be an NVAE adversary with resources bounded by $(t, \mathbf{q_e}, \mathbf{q_d}, \boldsymbol{\sigma})$. We have that*

$$\begin{aligned}\mathbf{Adv}^{\mathbf{nvae}(\tau_c)}_{\mathrm{OCBv}[E]}(\mathscr{A}) \leq \mathbf{Adv}^{\widetilde{\mathbf{sprp}}}_{E}(\mathscr{B}) &+ \sum_{\tau \in \mathcal{I}_T} \mathbf{Adv}^{\widetilde{\mathbf{sprp}}}_{E}(\mathscr{C}_\tau) \\ &+ \mathbf{Adv}^{\widetilde{\mathbf{sprp}}}_{E}(\mathscr{C}_{\tau_c}) + \frac{28.5q^2}{2^n} + \boldsymbol{q}_d[\tau_c]\frac{2^{n-\tau_c}}{2^n - 1}.\end{aligned}$$

for some $\mathscr{B}$ that runs in time $t+\beta \cdot q$ and makes $2q$ queries, and some $\mathscr{C}_\tau$ that runs in time $t+\gamma_\tau \cdot \boldsymbol{q}[\tau]$ and makes $2\boldsymbol{q}[\tau]$ queries for $\tau \in \mathcal{I}_T$, where $\boldsymbol{q}[\tau] = \lceil \boldsymbol{\sigma}[\tau]/n \rceil + 2 \cdot (\boldsymbol{q}_e[\tau] + \boldsymbol{q}_d[\tau])$ for $\tau \in \mathcal{I}_T$, and $q = \sum_{\tau \in \mathcal{I}_T} \boldsymbol{q}[\tau]$, and β and γ_τ for $\tau \in \mathcal{I}_T$ are constants.

Proof. The result in Theorem 8.15 follows from Theorem 8.11 and Lemma 8.14. The fraction $(28.5q^2)/(2^n)$ upper bounds the sum of all the terms that arise from all applications of Lemma 8.14. □

If we further make the reasonable assumption that all the adversaries are "optimal", i.e. the advantage $\mathbf{Adv}^{\widetilde{\mathbf{sprp}}}_{E}(\mathscr{C}_\tau)$ does not decrease when we increase $\boldsymbol{q}[\tau]$, then we can further simplify the bound to the form

$$\mathbf{Adv}^{\mathbf{nvae}(\tau_c)}_{\mathrm{OCBv}[E]}(\mathscr{A}) \leq (|\mathcal{I}_T| + 2) \cdot \mathbf{Adv}^{\widetilde{\mathbf{sprp}}}_{E}(\mathscr{B}) + \frac{28.5q^2}{2^n} + \boldsymbol{q}_d[\tau_c] \cdot \frac{2^{n-\tau_c}}{2^n - 1}.$$

Performance of OCBv The performance of OCBv can be expected to be very similar to that of OCB, as the two schemes only differ in the way the masking Δ-values are computed. In addition to the operations necessary to compute Δ-offsets in OCB, the computation of the L_τ-values has to be done for OCBv. However, these can be precomputed at the initialization phase and stored, so the cost of their computation will be amortized over all queries. The only additional processing that remains after dealing with L_τ-s is a single xor of a precomputed L_τ to a Δ-value, necessary in every query. This is unlikely to impact the performance significantly.

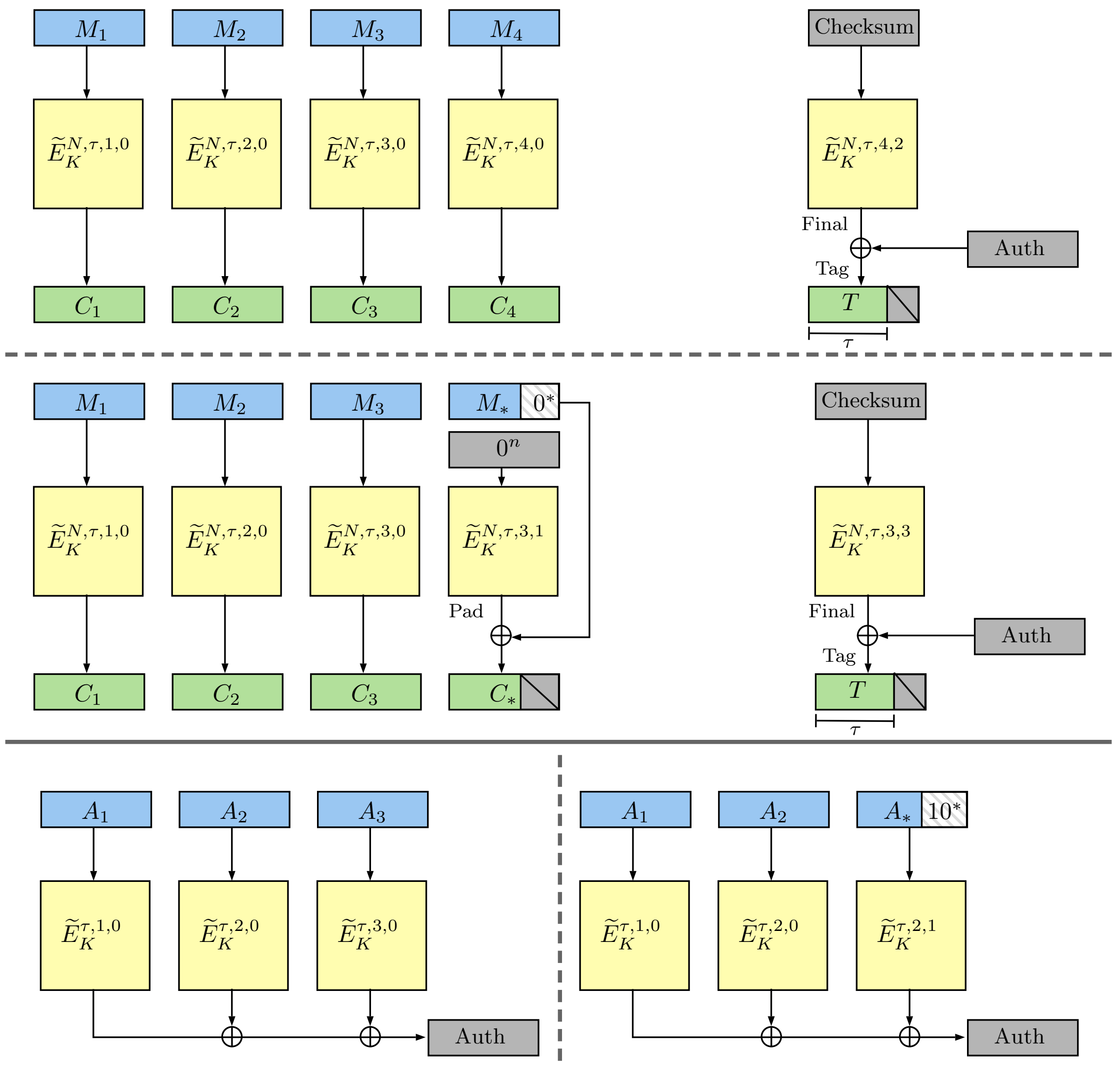

M_1
M_2
M_3
M_4
Checksum
$\widetilde{E}_K^{N,\tau,1,0}$
$\widetilde{E}_K^{N,\tau,2,0}$
$\widetilde{E}_K^{N,\tau,3,0}$
$\widetilde{E}_K^{N,\tau,4,0}$
$\widetilde{E}_K^{N,\tau,4,2}$
Final
Auth
Tag
C_1
C_2
C_3
C_4
T
τ
M_1
M_2
M_3
M_*
0^*
0^n
Checksum
$\widetilde{E}_K^{N,\tau,1,0}$
$\widetilde{E}_K^{N,\tau,2,0}$
$\widetilde{E}_K^{N,\tau,3,0}$
$\widetilde{E}_K^{N,\tau,3,1}$
$\widetilde{E}_K^{N,\tau,3,3}$
Pad
Final
Auth
Tag
C_1
C_2
C_3
C_*
T
τ
A_1
A_2
A_3
$\widetilde{E}_K^{\tau,1,0}$
$\widetilde{E}_K^{\tau,2,0}$
$\widetilde{E}_K^{\tau,3,0}$
Auth
A_1
A_2
A_*
10^*
$\widetilde{E}_K^{\tau,1,0}$
$\widetilde{E}_K^{\tau,2,0}$
$\widetilde{E}_K^{\tau,2,1}$
Auth

Figure 8.13 – **Illustration of the encryption algorithm of ΘCBv** (inspired by the original illustration for OCB3 [KR11]) instantiated with a tweakable blokcipher $\widetilde{E} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$. The top part depicts the encryption of a message with four complete blocks (top) with Sum= $\bigoplus_{i=1}^{4} M_i$ and the encryption of a message with three complete blocks and an incomplete block (bottom) with Sum= $\bigoplus_{i=1}^{3} \oplus M_* \| 10^*$. The bottom part of the picture shows processing of associated data of three complete blocks (left) or two complete blocks and an incomplete block (right).

Part III

Cryptanalysis

Chapter 9

Robustness of 3rd Round CAESAR Candidates to Nonce Reuse and Key Overuse

In this chapter, we switch from definitional work and design to cryptanalysis. We target the 3rd round CAESAR candidates, and the AE schemes CCM and GCM. For each of these, we attempt to find the most efficient nonce-reusing and/or high data-complexity decryption and/or forgery attack to determine to what degree they actually resist to attackers.

The work presented in this chapter is a result of joint work with Serge Vaudenay. It is available in IACR ePrint archive [VV17].

Organization of the Chapter. We briefly address related work in Section 9.1 and discuss the contributions in Section 9.2.

We give a motivation for finding attacks on all candidates in Section 9.3 and describe our attack model in Section 9.4. We summarize the results in Section 9.5.

In Section 9.6, we describe attacks that each apply to several AE schemes that share some common design element. We then give dedicated attacks on GCM and some of the 3rd round CAESAR candidates in Sections 9.7 to 9.19.

In Appendix C, we give a brief description of each scheme analysed in this chapter.

9.1 Related Work

The (in)security of GCM mode was investigated in a number of works [IOM12, Saa12, HP08, PC15], in particular Joux authored the “forbidden” nonce misusing attack [Jou06]. Collision attacks similar to ours, or inspiring ours, were described for previous versions of AEZ by Fuhr et al. [FLS15], and Chaigneau and Gilbert [CG16]. Collision attacks on OCB were given by Ferguson [Fer02] and Sun et al. [SWZ13]. Reusable forgery attacks on OCB, OTR and COLM were described by Forler et al. [FLLW17]. Collision-based attacks on COPA and ELmD (the predecessors of COPA) were described by Bay et al. [BEK16] and Lu [Lu17]. Bost and Sanders found a flaw in the masking scheme of an earlier version of OTR [BS16], Huang and Wu described a collision-based forgery attack on the same scheme [HW14]. Mileva et al. describe a nonce misusing distinguisher attack for MORUS [MDV16]. The collision-based forgeries on NORX, Ascon and Keyak described in this chapter are matching Lemma 2 of the work on provable generic security of full-state keyed duplex by Daemen et al. [DMV17]. The possibility of a low-complexity nonce reusing attack is mentioned in the AEGIS v1.1 specifications [WP16]. The designers of Ketje point at the possibility of nonce-reusing key recovery [BDP+16a]. Kales, Eichlseder and Mendel independently mounted state-or-key recovery attacks on Tiaoxin, AEGIS and MORUS with similar complexities our attacks [KEM17].

9.2 Contribution

The work presented in this chapter may be seen as a hybrid of a survey and cryptanalysis: our goal is to assemble an overview of attacks on all 3rd round CAESAR candidates, but we describe many of those attacks ourselves, whenever there were no attacks existing previously.

A collection of concrete attacks gives a tangible assessment of the actual impact of nonce-reuse/high-data-complexity attacks on the individual candidates, and provides extra information on top of the guarantees provided by the schemes' designers.

We show that the resilience of schemes with similar security claims can vary greatly. This can be useful to break ties at the end of the 3rd round of CAESAR competition. Some of these attacks, especially the low-complexity nonce-misuse attacks, can be viewed as disturbingly powerful. Consequently, some of the candidates may be revealed to be too brittle for general-purpose use.

The collection of generic attacks in Section 9.6 also helps to identify common security phenomena related to similar construction principles shared by certain candidates.

A remark on the selection of CAESAR finalists. During the 2nd round, the CAESAR committee has introduced the *use cases* [Ber16]. These were supposed to represent the main types of application of AE schemes, each of them loosely describing a set of design goals. Each candidate was then supposed to specify which use case(s) does it target. The proposed use cases were (1) “Lightweight applications” for resource constrained environments, (2) “High-performance applications” for high-performance software implementation on general-purpose CPUs, and (3) “Defense in depth” for AE robust to various kinds of misuse. In particular, “robustness” was only explicitly required in use case (3).

In March 2018, the 3rd round of CAESAR finished, and 7 finalists were announced.[1]

[1]We note that the *final portfolio* was yet to be announced at the time of writing of this dissertation,

Three of the finalists, namely ACORN, AEGIS and MORUS, succumb to low-complexity key-or-state recovery attacks that fit well within the scope of accidental nonce reuse, as shown in this chapter and independently by Kales, Eichlseder and Mendel [KEM17].

Although none of these three candidates targeted use case (3), the author of this dissertation is not certain whether they should have been included among the finalists. While it is understood that the use cases make it clear that only the schemes targeting the case (3) can be expected to possess any kind of robustness, promoting schemes that possess *absolutely no robustness* against nonce reuse may be dangerous. Not just because of users who may use one of the three brittle candidates incorrectly, but also because the schemes in the final portfolio will become a reference for future designs. And ACORN, AEGIS and MORUS will be setting the bar very low with respect to resistance to key recovery attacks.

9.3 Motivation

Complementing provable security. The results presented in the previous chapters were done in the spirit of provable security. We either worked towards achieving some formal definitions of AE security when designing AE schemes, or we sought to capture some novel security properties in new definitions. Targeting a properly defined security notion when designing an AE scheme has many benefits. It reassures both the designer and the user that the scheme in question will (very likely) deliver on some meaningful, and typically pretty strong set of security properties. It is also made quite clear when do these guarantees apply, and when do they become void.

Take a nonce-based AE scheme that comes with a proven NAE security bound (see Definition 2.8) as an example. The user knows that if the scheme is used correctly, the produced ciphertexts will look close to uniform strings to third parties, and that it is very unlikely that an adversary will be able to forge any ciphertext. What "use correctly" means here is "do not repeat nonces, and do not process too much data," the amount being implied by the security bound.[2] As soon as any of these usage conditions is violated, the user should assume that all the security is forfeit. This is the conservative practice that any user should adopt.

Yet, the security claims related to some well-established notion give no indication about what happens *after* the guarantees are void. This is information that may not be useful for all the users of AE, but it may be of value to experienced security engineers or members of standardisation (or competition) committees. For example, when choosing between two nonce-based AE schemes Π_1 and Π_2 with similar performance on the target platform, similar memory footprint and similar NAE security bounds, the knowledge that Π_2 succumbs to a key recovery when nonces repeat, while for Π_1 only existential forgery

and may not contain all the 7 finalists.

[2]We note that the meaning of the phrase "use correctly" is in reality much more complicated, and possibly not yet completely understood. Nevertheless, we simplify its meaning to keep the discussion focused.

attacks are known under the same setting, will make the choice rather easy.

In this chapter, we aim at providing fine-grained information beyond what is implied by the security claims made by the 3rd round CAESAR candidates. Our method is to analyse each candidate, CCM and GCM, and report the best attacks we can find. The type of attack, its complexity and adversarial powers necessary to break each candidate will then shed more light on its actual resilience.

64-bit bound and nonce-misuse. The main goal of the CAESAR competition was set to "identify a portfolio of authenticated ciphers that offer advantages over AES-GCM and are suitable for widespread adoption" [Ber14a]; GCM instantiated with the AES blockcipher was taken to be a reference that ought to be surpassed by the CAESAR candidates, while the name of the competition spells out the properties the candidates are expected to guarantee: security, applicability and robustness.

For this chapter, we take the liberty to interpret robustness of AE schemes as the ability to resist powerful attacks, possibly beyond the limitations imposed by their designers. We focus on nonce reuse and high data complexity, simply because every candidate must make a claim about security with respect to these specific usage conditions.

Only three candidates guarantee security beyond nonce reuse. AEZ and Deoxys II guarantee no degradation of authenticity, and the minimal (and unavoidable [RS06b]) degradation of confidentiality even if the nonces are repeated. COLM guarantees a weaker version of confidentiality protection in the presence of nonce misuse, OAE1 [FFL12] security.

Concerning the data that can be processed with a single key, most CAESAR candidates guarantee security up to the so called birthday-bound; for AES-based AE schemes, this means processing no more than about 2^{64} blocks of data per key and making no more than 2^{64} encryption queries. In this chapter, we use the 64-bit data/query complexity as a reference threshold for comparison of candidates, denoted by 64-bit-bound.

In Table 9.1, we categorize the 3rd round candidates, as well as CCM and GCM, *based on their security claims* with respect to the nonce misuse and quantitative security. We consider a scheme to claim security against nonce reuse if it claims security in the sense of MRAE [RS06b], OAE1 [FFL12] or RAE [HKR15]. For each candidate, we consider an instance with a 128-bit secret key.

9.4 Notation and Security Model

When presenting the CAESAR candidates, we try to respect the original notations but deviate a bit to unify the notation of the common input/output values. Hence, the secret key is denoted by K, the nonce (or IV) is denoted by N, the associated data (AD) is denoted by A, the plaintext is denoted by M, the ciphertext is denoted by C, and the tag (if any) is denoted by T. We further use τ to denote the ciphertext expansion/stretch, which is in most cases the same as the tag length.

	up to 64-bit-bound	**beyond 64-bit-bound**
unique nonces	OCB, NORX, Jambu, CLOC&SILC	Tiaoxin, Morus, Keyak, Ketje, Deoxys I&II, Ascon, AEGIS, ACORN
nonce misuse	Deoxys II, COLM, AEZ	-

Table 9.1 – **An overview of 3rd round CAESAR candidates based on their *claimed* security guarantees** w.r.t the nonce misuse and quantitative security; 64-bit-bound refers to about 2^{64} processed bits. For security in presence of nonce misuse, we consider MRAE [RS06b], OAE [FFL12] or RAE [HKR15]. For each candidate, we consider an instance with 128-bit secret key.

Each of the candidates internally partitions the inputs into blocks of constant size. We use several symbols to denote the length of the blocks, e.g. n, r or ν, in order to respect the notation of each candidate as much as possible. We use subscript to index blocks in a query and superscript to index queries, e.g. M_i^j is the i^{th} message block in j^{th} query.

With a slight abuse of notation, we let $X0^*1$ denote extending a string X with the smallest number of zero bits followed by a “1” that will yield a string whose length is a multiple of the block size, when a block size is implicit from the context. For blockcipher-based schemes, we let E denote the underlying blockcipher. We let $\|_{i=1}^{j} M_i$ denote the concatenation $M_1\|\ldots\|M_j$ for any $M_1,\ldots,M_j \in \{0,1\}^*$.

Syntax. In this chapter, each CAESAR candidate is viewed as nonce-based AE scheme (see Section 2.4). We note that most of the candidates internally compute an encryption of the message C and an authentication tag T, and output $C\|T$ as the final ciphertext. We refer to C as the core ciphertext and call T authentication tag, or simply tag.

Attack model. We focus on three types of attacks: decryption attacks, (semi) universal forgeries and key recovery attacks. To make the results comparable, for each candidate we attack an instance that uses 128-bit keys (i.e. $\mathcal{K} = \{0,1\}^{128}$), and we define our attack models to correspond to the 128-bit security level.

In each attack on a scheme Π, an attacker $\mathscr{A}$ has blackbox oracle access to an instance of the encryption and the decryption algorithms $\mathcal{E}_K, \mathcal{D}_K$ of Π that use a secret key K unknown to $\mathscr{A}$. We call $\mathscr{A}$ *nonce respecting* if each encryption query it makes uses a distinct nonce. We say that $\mathscr{A}$ mounts a *chosen plaintext attack* (CPA) if it never makes a decryption query, otherwise we say $\mathscr{A}$ mounts a *chosen ciphertext attack* (CCA).[3]

For each attack, we keep track of the data complexity (in blocks of some constant size) and/or the query complexity, and of the maximal number of encryption queries that reuse the same (but arbitrary) nonce.

[3]Note that this distinction does not make sense for forgeries.

(Semi)-universal forgery. $\mathscr{A}^{\mathcal{E}_K,\mathcal{D}_K}(N, A, M)$ receives a nonce, AD and a message and tries to produce a decryption query (N, A, C) that will correctly decrypt to M, such that C was not an output of a previous encryption query made with N, A. We call the forgery *semi-universal* if the adversary only gets target AD and message (i.e. $\mathscr{A}^{\mathcal{E}_K,\mathcal{D}_K}(A, M)$) or only a target message (i.e. $\mathscr{A}^{\mathcal{E}_K,\mathcal{D}_K}(M)$) and is allowed to use arbitrary values for the remaining inputs.

Decryption attack. $\mathscr{A}^{\mathcal{E}_K,\mathcal{D}_K}(N, A, C)$ receives a nonce, AD and ciphertext-tuple that is an encryption of a secret random message M of fixed length $\mu \geq 128$, and tries to produce M.

Key recovery. $\mathscr{A}^{\mathcal{E}_K,\mathcal{D}_K}()$ tries to compute K.

Reusable attacks We call a forgery (resp. decryption) attack *reusable* if, after having forged (resp. decrypted) for the first time, the query and computational complexity of the consequent forgeries (resp. decryptions) are "significantly lower" than the complexity of the initial forgery (resp. decryption).

9.5 Results

We sort the CAESAR candidates into six categories based on the adversarial powers necessary to break them: **(A)** Those for which we have a nonce-respecting universal forgery *and* a decryption attack at the 64-bit-bound. **(B)** Those others for which we have a nonce-respecting universal forgery *and* a decryption attack beyond the 64-bit-bound, but below exhaustive search. **(C)** Those for which we have a reusable forgery *and* a reusable decryption attack with small complexity, possibly with nonce-misuse. **(D)** Those others for which we have a forgery *or* a decryption attack with small complexity, possibly with nonce-misuse. **(E)** Those others for which we have a forgery *or* a decryption attack at the 64-bit-bound, possibly with nonce-misuse. **(F)** Remaining ones.

Our results are summarized in Table 9.2, where the categories (A), (B), (C), (D), (E) and (F) are listed in this order. For each candidate, we indicate the type of attack, the query complexity[4], whether the attack needs nonce misuse, and whether it is reusable. The comments "(N, A)", "(N)" and "(A)" mean that the reusability is limited to a fixed pair of nonce and AD, a fixed nonce or a fixed AD, respectively. All attacks presented in Table 9.2 succeed with high probability.

The categories can be ordered by a decreasing level of resilience: (F)≥(E)≥(D)≥(C) and (F)≥(E)≥(B)≥(A). The categories (A) and (C) are incomparable (same for (B) and (D)), as the impacted schemes succumb to different kinds of misuse. However, the attacks in category (C) may be seen as a more serious threat than those in category (A), as they are much more likely to occur in practice.

[4]The time and memory complexities of the attacks mentioned in the Table 9.2 are small multiples/small powers of the query complexity.

	algorithm	source(s)	type of attack	nonce-reuse	# queries	reusable
A	**AES-GCM**$^{\#}$ [MV04]	9.7	univ. forgery	1	$3 \cdot 2^{64}$	yes
	AEZ [HKR17]	9.8, [CG16]	key recovery	1	$3 \cdot 2^{64}$	
	OCB [KR16]	9.9, [Fer02]	univ. forgery & CCA decryp.	1	2 (one w/ 2^{64} blocks)	yes
	AES-OTR [Min16]	9.6, 9.10	univ. forgery & CPA decryp.	1	2 (one w/ 2^{64} blocks)	yes
B	CLOC [IMG+16]	9.11	univ. forgery & CPA decryp.	1	2^{80}	yes
C	**AES-GCM** [MV04]	9.6, 9.7, [Jou06]	univ. forgery & CPA decryp.	2	2	yes
	DEOXYS-I [JNP16]	9.6	univ. forgery & CCA decryp.	3	3	yes (A)
	OCB [KR16]	9.6	univ. forgery & CCA decryp.	2	2	yes (A)
	Tiaoxin [Nik16]	9.13	key recovery	30	30	
	AEGIS-128 [WP16]	9.14	univ. forgery & CPA decryp.	15	15	yes (N, A)
	ACORN-128 [Wu16]	9.15	univ. forgery & CPA decryp.	586	586	yes (N, A)
	Ketje Sr [BDP+16a]	9.16	key recovery	50	50	
	MORUS 640 [WH16a]	9.17	univ. forgery & CPA decryp.	8	8	yes (N)
D	**AES-CCM** [WHF03b]	9.6	CPA decryp.	2	1	
	CLOC & SILC [IMG+16]	9.6	CPA decryp.	2	1	no
	JAMBU [WH16b]	9.6	CPA decryp.	$1 + \lvert C \rvert/64$	$\lvert C \rvert/64$	no
	NORX32-4-1 [AJN16]	9.6	CPA decryp.	$1 + \lvert C \rvert/384$	$\lvert C \rvert/384$	no
	Ascon-128 [DEMS16]	9.6	CPA decryp.	$1 + \lvert C \rvert/64$	$\lvert C \rvert/64$	no
	Lake Keyak [BDP+16b]	9.6	CPA decryp.	$1 + \lvert C \rvert/1344$	$\lvert C \rvert/1344$	no
E	COLM [ABD+16]	9.6	semi-univ. forgery	$1 + q$	2^{64}	yes (N, A)
F	Deoxys-II [JNP16]	9.19	semi-univ. forgery & CCA decryp.	2^m	2^{128-m}	yes (A)

Table 9.2 – A summary of attacks on 3rd round CAESAR candidates and their clustering based on the type of attack. The categories (A), (B), (C), (D), (E) and (F) are listed from top to bottom. The column "source" lists the sections and/or bibliography references that describe the relevant attacks. The comments "(N, A)", "(N)" and "(A)" in the reusability column (see Section 9.4) mean that the reusability is limited to fixed values of the listed parameters. The values in the column "nonce-reuse" indicate maximal number of times any nonce is used (so 1 means nonce respecting), q denotes the number of independent forgeries made in a single attack, and m is used as a parameter.
$^{\#}$The attack applies only if $|N| > 128$.

Disclaimer. We understand that **none** of the attacks we present violates the security claims of any of the CAESAR candidates. That is not the goal of our work. Our goal is to determine to what degree will the security of respective candidates deteriorate *after* the guarantees become void. Each of the attacks we present breaks the usage conditions imposed by the authors of the corresponding candidate, either by reusing the nonces, or by processing too much data.

9.6 Generic Attacks

In this section, we list attacks that trivially apply to certain construction principles, rather than being construction-specific.

CPA decryption: streamciphers (nonce reuse, constant complexity) candidates that produce a core ciphertext C and a tag T such that $C = M \oplus f(K, N, |M|)$ (or $C = M \oplus f(K, N, A, |M|)$), i.e. the message is xored with a sequence of masking bits derived as a function of the nonce and the secret key (or the nonce, secret key and AD) will necessarily succumb to this attack. To decrypt $(N, A, C\|T)$, we make a single encryption query $f(K, N, A, |M|)\|T' = \mathcal{E}_K(N, A, 0^{|C|})$ that reveals the key stream and compute $M = C \oplus f(K, N, A, |M|)$. This attack applies to **CCM**, **GCM**.

CPA decryption: self-synchronizing streamciphers (nonce reuse, tiny complexity) The previous attack can be adapted to AE schemes that produce the core ciphertext C block by block, by xoring the current message block with masking bits dependent on the key, the nonce, AD and the previous message blocks. I.e. given a partitioned message $M_1, \ldots, M_\ell \overset{n}{\leftarrow} M$ they compute the i^{th} block of ciphertext as $C_i = M_i \oplus f(K, N, A, M_1\| \ldots M_{i-1}, |M_i|)$ (where the value of n depends on the scheme).

To mount a decryption attack with an input tuple $(N, A, C\|T)$, we make $|C|_n$ encryption queries as follows:

```
1: Compute $C_1, \ldots, C_\ell \overset{n}{\leftarrow} C$.
2: for $i \leftarrow 1$ to $\ell$ do
3:     Query $C'\|T' \leftarrow \mathcal{E}_K(N, A, M_1\| \ldots \|M_{i-1}\|0^{|C_i|})$.
4:     Compute $C'_1, \ldots, C'_i \overset{n}{\leftarrow} C'$ and then $M_i \leftarrow C'_i \oplus C_i$.
5: end for
6: return $M_1\| \ldots \|M_\ell$
```

This attack applies to **CLOC**, **SILC**, **AEGIS**, **ACORN**, **MORUS**, **Ketje**, **NORX**, **Ascon**, **Keyak** and **JAMBU**.

Semi-universal forgery: AD preprocessing (nonce reuse, varying complexity). Several candidates internally process an encryption query $\mathcal{E}(K, N, A, M)$ by first computing a value $V = f(K, N, A)$ dependent on the key, nonce and the AD, and then compute the (tagged) ciphertext as a function of the secret key, the message and the value V as $C = g(K, V, M)$, such that $|V| = v$ for constant v. If $|\mathcal{N}| \geq 2^{v/2}$, then it is possible to find a pair $(N_1, A_1), (N_2, A_2)$ such that $f(K, N_1, A_1) = f(K, N_2, A_2)$ in a nonce-respecting birthday attack, and then use it to forge a ciphertext for a challenge message M (hence this yields a semi-universal forgery):

```
1: Initialize empty table $\mathsf{T}$, pick arbitrary $\hat{M} \in \{0,1\}^{2v}$.
2: for $i \leftarrow 1$ to $2^{v/2}$ do
3:     Pick $(N', A')$ with a fresh $N'$ randomly.
4:     Query $C' \leftarrow \mathcal{E}_K(N', A', \hat{M})$, then insert $(C', (N', A'))$ to $\mathsf{T}$.
5: end for
6: Find entries $(C', (N_1, A_1))$, $(C', (N_2, A_2))$ (with collision on $C'$) in $\mathsf{T}$.
7: Query $C \leftarrow \mathcal{E}_K(N_1, A_1, M)$ and forge with $(N_2, A_2, C)$.
```

The attack succeeds with a probability close to 1/2, in particular choosing $\hat{M} \in \{0,1\}^{2v}$ ensures that a C' collision implies a V collision with overwhelming probability. It is reusable with the same $(N_1, A_1), (N_2, A_2)$, and uses every nonce no more than $1 + q$ times, with q the number of desired forgeries.

The attack applies with 64-bit-bound complexity (as $v = 128$) to, **AEZ**, **CLOC**, **SILC**, **COLM** and with some care to **CCM**.[5]

Semi-universal forgery: sponges (nonce reuse, varying complexity). In all sponge-based modes, the processing can again be expressed with two functions f and g in a similar way as in the previous attack on AD preprocessing. However, nonce reuse allows the attacker to force an arbitrary value to r bits of the sponge state after the processing the first message block. The processing of an encryption query $\mathcal{E}(K, N, A, M)$ by a sponge-based scheme can be modelled as follows: first partition the message $M_1, \ldots, M_\ell \xleftarrow{r} M$, then compute $V = f(K, N, A)$ with $|V| = b$, then compute the first cipertext block as $C_1 = \mathsf{left}_r(V) \oplus M_1$, and compute the rest of the tagged ciphertext as $C_2 \| \ldots \| C_\ell \| T = g(K, V \oplus (M_1 \| 0^c), M_2 \| \ldots \| M_\ell)$, where $c = b - r$.

Using this, the previous attack can be adapted to work with query complexity $2^{c/2}$ (where c is the capacity of the given sponge-based scheme) to forge a ciphertext for arbitrary (A, M):

```
1: Initialize empty table $\mathsf{T}$, pick arbitrary $\hat{M} \in \{0,1\}^c$.
2: for $i \leftarrow 1$ to $2^{c/2}$ do
3:     Pick a fresh $N'$ randomly.
4:     Query $C' \| T' \leftarrow \mathcal{E}_K(N', A, 0^r)$, then query $C'' \| T'' \leftarrow \mathcal{E}(N', A, C' \| \hat{M})$.
5:     Compute $C''_1, \ldots, C''_\ell \xleftarrow{r} C''$, then insert $(C''_2 \| \ldots \| C''_\ell \| T'', (N', C'))$ to $\mathsf{T}$.
6: end for
7: Find entries $(C'' \| T'', (N_1, C_1))$, $(C'' \| T'', (N_2, C_2))$ (with collision on $C'' \| T''$) in $\mathsf{T}$.
8: Query $C \| T \leftarrow \mathcal{E}_K\left(N_1, A, M \oplus ((C_1 \oplus C_2) \| 0^{|M|-r})\right)$ and forge with $(N_2, A, C \| T)$.
```

The success probability is close to $1/2$. The second query in the attack forces the internal state of the sponge to become $0^r \| S$ for some $S \in \{0,1\}^c$, hence the birthday complexity in c. We xor the difference of C_1 and C_2 to the first block of M to force the repetition of the r outer bits of the state during forgery.

The attack is reusable with the same $(N_1, A), (N_2, A)$,[6] and uses every nonce no more than $2 + q$ times, with q the number of desired forgeries.

The attack applies with 64-bit-bound complexity (as $c = 128$) to **NORX** and with beyond 64-bit-bound complexity (as $c = 256$) to **Keyak** and **Ascon**. We note that for Keyak and Ascon, the exhaustive key search has the same time complexity as this attack, but needs only a single query.

Universal forgery and CCA decryption: ciphertext translation (nonce misuse, tiny complexity). Some candidates use so called *ciphertext translation* [Rog02] to incorporate the authentication of AD with a message-only encryption core $\bar{\mathcal{E}}$. The method is described in Figure 8.1. It can be briefly described as follows. Schemes based

[5]For CCM with $\tau = 128$, we must use A' of 240 bits to make sure that the encoding of the nonce and AD for the CBC MAC is block-aligned.

[6]For Keyak, the attack can be reused with arbitrary AD, because it processes AD and message simultaneously.

on ciphertext translation compute the tagged ciphertext as

$$\mathcal{E}_K(N, A, M) = \bar{\mathcal{E}}_K(N, M) \oplus 0^{|M|} \| H_K(A)$$

where $\bar{\mathcal{E}}_K(N, M)$ returns a core-ciphertext and a τ-bit tag and H is a keyed function with τ-bit output. To forge for (N, A, M), we pick arbitrary $\hat{N} \neq N$, $\hat{M} \neq M$ and $A' \neq A$ and we do:

```
1: Query $C^1 \| T^1 \leftarrow \mathcal{E}_K(\hat{N}, A, \hat{M})$ and $C^2 \| T^2 \leftarrow \mathcal{E}_K(\hat{N}, A', \hat{M})$.
2: Compute $\Delta \leftarrow T^1 \oplus T^2$.
3: Query $C' \| T' \leftarrow \mathcal{E}_K(N, A', M)$ and forge with $(N, A, C' \| (T' \oplus \Delta))$.
```

It is easily verified that the forgery is correct. This attack can be modified to decrypt a ciphertext $N, A, C \| T$; knowing Δ, we query $N, A', C \| (T \oplus \Delta)$ and learn the message M. This attack applies to **OCB**, **AES-OTR** and **Deoxys-I**.

9.7 AES-GCM

A brief description of GCM can be found in Appendix C.2.

Universal forgery (nonce misuse, tiny complexity). This attack was first described by Joux as the "forbidden attack" [Jou06]. The main idea is that recovering the authentication key L makes forging very easy. We assume that $\tau = 128$. To forge for N, A, M, we pick random $\bar{N}$ and $M^1 \neq M^2 \in \{0,1\}^{128}$ and do:

```
1: Query $C^1 \| T^1 \leftarrow \mathcal{E}_K(N, \varepsilon, M^1)$ and $C^2 \| T^2 \leftarrow \mathcal{E}_K(N, \varepsilon, M^2)$.
2: Compute $L$ as root of $P(\Lambda) = (C_1^1 \oplus C_1^2) \cdot \Lambda^2 \oplus (T^1 \oplus T^2)$ over $\mathsf{GF}(2^{128})$.
3: Query $C' \| T' \leftarrow \mathcal{E}_K(N, A', M')$ with arbitrary $A'$ and $M'$ s.t. $|M'| = |M|$.
4: Forge with $(N, A, (C' \oplus M' \oplus M) \| (T' \oplus \mathrm{GHASH}_L(A', C') \oplus \mathrm{GHASH}_L(A, C)))$.
```

We note that L will be the only root of $P(\Lambda)$ as squaring yields a bijection over $\mathsf{GF}(2^{128})$. Once L is computed, forgeries become easy.

Universal forgery (nonce respecting, 64-bit-bound, $|N| > 128$). If nonces longer than 128 bits are allowed, it is possible to recover the authentication key L in a nonce-respecting birthday attack. We note, however, that the use of nonce length other than 96 bits is uncommon and discouraged [IOM12]. Assuming that $\tau = 128$, for each i we use distinct N^i of 256 bits and $M^i = B \| M_2^i$ for a fixed $B \in \{0,1\}^{128}$ and distinct $M_2^i \in \{0,1\}^{128}$, and do:

```
1: for $i \leftarrow 1$ to $2^{64}$ do query $C^i \| T^i \leftarrow \mathcal{E}_K(N^i, \varepsilon, M^i)$.
2: For $i \neq j$ s.t. $C_1^i = C_1^j$ find $L$ as root of $P(\Lambda) = (C_2^i \oplus C_2^j) \cdot \Lambda^2 \oplus (T^1 \oplus T^2)$.
3: Forge using $L$.
```

Note that the collision in line 2 must imply $\mathrm{GHASH}_L(\varepsilon, N^i) = \mathrm{GHASH}_L(\varepsilon, N^{i'})$, so if it occurs, the attack succeeds. We note that a forgery allows to mount a similar CCA decryption attack as for the ciphertext translation-based AE schemes (by changing AD and the tag accordingly).

9.8 AEZ v5

We present three nonce-respecting attacks that respectively recover the subkeys I, J and L, each at the 64-bit-bound complexity. A brief description of AEZ is given in Appendix C.3. The same appendix explains the notations E and AES4.

***J*-recovery attack.** The Chaigneau-Gilbert attack [CG16] on AEZ v4.1 can be applied to AEZ v5 to extract J by a nonce-respecting chosen message attack at the birthday bound. When N and A are single blocks, then based on the AEZ v5 specification [HKR17] the function H becomes

$$\begin{aligned} h_k(\tau, N, A) &= E_K^{3,1}(\tau) \oplus E_K^{4,1}(N) \oplus E_K^{5,1}(A) \\ &= E_K^{3,1}(\tau) \oplus \mathsf{AES4}_k(N \oplus 4J \oplus 2I \oplus L) \oplus \mathsf{AES4}_k(A \oplus 5J \oplus 2I \oplus L). \end{aligned}$$

If we limit ourselves to queries with $A = N \oplus c$ for a fixed block c and variable nonces, a ciphertext collision with the pair (N, N') will mean that $N' = N \oplus c \oplus J$. The attack runs as follows:

```
1: Initialize an empty table $\mathsf{T}$.
2: Pick an arbitrary block $c \in \{0,1\}^{128}$ and message $M \in \{0,1\}^{2\cdot 128}$.
3: for $i \leftarrow 1$ to $2^{64}$ do
4:     Pick a fresh $N$ randomly, set $A \leftarrow N \oplus c$.
5:     Query $C \leftarrow \mathcal{E}_K(N, A, \tau, M)$, store $(C, N)$ in $\mathsf{T}$.
6: end for
7: Find $(C, N), (C', N')$ in $\mathsf{T}$ with $C = C'$, compute $J = N \oplus N' \oplus c$.
```

The Chaigneau-Gilbert attack requires a little effort to be adapted to AEZ v5 but it can recover I and L with nonce-misuse. A nonce respecting recovery of I and L is possible if we can use nonces of several blocks (a feature of AEZ [HKR17]), to have a similar attack as the one above.

***L*-recovery attack.** If $|N|_{128} = 2$ and $A = \varepsilon$, then following the AEZ v5 specifications the function H becomes

$$\begin{aligned} h_k(\tau, (N_1, N_2)) &= E_K^{3,1}(\tau) \oplus E_K^{4,1}(N_1) \oplus E_K^{4,2}(N_2) \\ &= E_K^{3,1}(\tau) \oplus \mathsf{AES4}_k(N_1 \oplus 4J \oplus 2I \oplus L) \oplus \mathsf{AES4}_k(N_2 \oplus 4J \oplus 2I \oplus 2L). \end{aligned}$$

We modify the J-recovery attack to use 2-block nonces with $N_2 = N_1 \oplus c$ for a fixed block c. A ciphertext collision with N and N' will then yield $L = N_1 \oplus N_1' \oplus c$:

```
1: Initialize an empty table $\mathsf{T}$.
2: Pick arbitrary block $c \in \{0,1\}^{128}$ and message $M \in \{0,1\}^{2\cdot 128}$.
3: for $i \leftarrow 1$ to $2^{64}$ do
```

```
4:     Pick a fresh $N_1$ randomly, set $N_2 \leftarrow N_1 \oplus c$.
5:     Query $C \leftarrow \mathcal{E}_K(N, \varepsilon, \tau, M)$, store $(C, N_1)$ in $\mathsf{T}$.
6: end for
7: Find $(C, N_1), (C', N_1')$ in $\mathsf{T}$ with $C = C'$, compute $L = N_1 \oplus N_1' \oplus c$.
```

Thus we recover L in a similar attack as before.

***I*-recovery attack.** Next, we see that when $|N|_{128} = 9$, the hash function H becomes

$$\begin{aligned} h_k(\tau, (N_1, \dots, N_9)) &= E_K^{3,1}(\tau) \oplus E_K^{4,1}(N_1) \oplus \cdots \oplus E_K^{4,9}(N_9) \\ &= E_K^{3,1}(\tau) \oplus \mathsf{AES4}_k(N_1 \oplus 4J \oplus 2I \oplus L) \oplus \cdots \oplus \\ &\quad \mathsf{AES4}_k(N_7 \oplus 4J \oplus 2I \oplus 7L) \oplus \mathsf{AES4}_k(N_8 \oplus 4J \oplus 2I) \oplus \\ &\quad \mathsf{AES4}_k(N_9 \oplus 4J \oplus 4I \oplus L). \end{aligned}$$

We again modify the J-recovery attack to use 9-block nonces with $N_2, \dots, N_8$ constant and $N_9 = N_1 \oplus c$ for a fixed block c. A ciphertext collision with N and N' yields $6I = N_1 \oplus N_1' \oplus c$:

```
1: Initialize an empty table $\mathsf{T}$.
2: Pick arbitrary $c \in \{0,1\}^{128}$, $N_2 \| \dots \| N_8 \in \{0,1\}^{7 \cdot 128}$ and message $M \in \{0,1\}^{2 \cdot 128}$.
3: for $i \leftarrow 1$ to $2^{64}$ do
4:     Pick a fresh $N_1$ randomly, set $N_9 \leftarrow N_1 \oplus c$.
5:     Query $C \leftarrow \mathcal{E}_K(N, \varepsilon, \tau, M)$, store $(C, N_1)$ in $\mathsf{T}$.
6: end for
7: Find $(C, N_1), (C', N_1')$ in $\mathsf{T}$ with $C = C'$, compute $I = (N_1 \oplus N_1' \oplus c) \cdot 6^{-1}$.
```

So, we recover I, J, L with a nonce-respecting chosen message attack at the 64-bit-bound. For AEZ, this is equivalent to recovering the secret key.

9.9 OCB3 (OCB v1.1)

We give a brief description of OCB in Appendix C.4.

***L*-recovery attack.** An attack by Ferguson [Fer02] allows to recover the derived key L. Essentially, it encrypts a very long random message and looks for collisions of the values $M_i \oplus C_i$. If such a collision occurs for $i \neq j$, we deduce $M_i \oplus \gamma_i \cdot L = M_j \oplus \gamma_j \cdot L$. Indeed, if $M_i \oplus \gamma_i \cdot L = M_j \oplus \gamma_j \cdot L$, we have $M_i \oplus \Delta_i = M_j \oplus \Delta_j$ so $M_i \oplus C_i = M_j \oplus C_j$. With this equation, we deduce L. This attack is nonce-respecting and works at the birthday bound.

Querying a huge message can be avoided in the nonce-misuse setting: make many queries (N, A^i, M) with $M = \varepsilon$ empty and $A^i = A_1^i \| A_1^i$ of two equal blocks. Then, if $A_1^i = A_1^j \oplus (\gamma_1 \oplus \gamma_2) \cdot L$ for some $i \neq j$, we observe a collision $T' = T$. This allows us to recover L.

Universal forgery (tiny complexity, using L). Using L, we can make a universal forgery for (N, A, M'). If $|M'|_{128} = \ell > 1$, we do:

```
1: Define a permutation $\pi : \{1, \dots, \ell\} \to \{1, \dots, \ell\}$ as $\pi(i) = (i + 1 \bmod \ell) + 1$.
2: for $i \leftarrow 1$ to $\ell$ do $M_i \leftarrow M'_{\pi(i)} \oplus \gamma_i \cdot L \oplus \gamma_{\pi(i)} \cdot L$.
3: Query $C \| T \leftarrow \mathcal{E}_K(N, A, M)$.
4: for $i \leftarrow 1$ to $\ell$ do $C'_i = C_{\pi^{-1}(i)} \oplus \gamma_i \cdot L \oplus \gamma_{\pi^{-1}(i)} \cdot L$.
5: Forge with $(N, A, C' \| T)$.
```

This attack is nonce-respecting and using a single encryption query, but needs L.

If $|M'|_{128} = 1$, we construct $M = M' \| (\gamma_1 \oplus \gamma_2) \cdot L$, make a query with (N, A, M) to get $C \| T$, and take $C' = C_1$, which again gives a valid encryption $C' \| T$ of (N, A, M').

***E_K* oracle (tiny complexity, using L).** We can also implement an E_K oracle, i.e. evaluate the underlying blockcipher on any plaintext. This can be used to e.g. mount universal forgeries or bootstrap the CCA decryption presented in this section. To compute $y_i = E_K(x_i)$ for arbitrary $x_1, \dots, x_s \in \{0,1\}^{128}$ set $\ell = 2^{14}$, and do:

```
1: Pick $M \in \{0,1\}^{\ell \cdot 128}$ with $\bigoplus_{i>1} M_i = (2^{-1} \oplus \gamma_1 \oplus \gamma_\ell) \cdot L$ randomly.
2: Query $C \| T \leftarrow \mathcal{E}_K(N, \varepsilon, M)$, compute $R \leftarrow C_1 \oplus T \oplus \gamma_1 \cdot L$.
3: Find $i$ s.t. $M_i \oplus R \oplus \gamma_i \cdot L = 0^7 \| 1 \| N'' \| 0^6$ for $N' \in \{0,1\}^{114}$.
4: Set $N' \leftarrow N'' \| 0^6$, compute $R' = C_i \oplus R \oplus \gamma_i \cdot L$.
5: for $i \leftarrow 1$ to $s$ do set $M'_i \leftarrow x_i \oplus R' \oplus \gamma_i \cdot L$.
6: Query $C' \| T' \leftarrow \mathcal{E}_K(N', \varepsilon, M')$.
7: for $i \leftarrow 1$ to $s$ do compute $y_i \leftarrow C'_i \oplus R' \oplus \gamma_i \cdot L$.
```

The R computed on line 2 is correct as $T = E_K(M_1 \oplus R \oplus \gamma_1 \cdot L) = C_1 \oplus R \oplus \gamma_1 \cdot L$. We can also add an unused nonce to the list of x_i-s to avoid making the $2^{14} \cdot 128$bit= 256KB query more than once. Then the attack uses a single encryption query per list of blocks $x_1, \dots, x_s$, of size $s + 1$ blocks.

CCA decryption attack (odd number of blocks, tiny complexity, using L). Assume that we want to decrypt (N, A, C, T) (let M be its decryption). We can first compute R associated with N with the above E_K oracle, as well as some fresh N' and its associated R' with tiny complexity. The tuple (N', A, M'), with the message M' defined by $M'_i = M_i \oplus R \oplus R'$, encrypts into (C', T') such that $C'_i = C_i \oplus R \oplus R'$ and $T' = T$ when ℓ is odd and the length of M is a multiple of the blocksize. So, a CCA decryption query with (N', A, C', T) gives M' from which we deduce $M = M' \oplus R \| \dots \| R$.

9.10 AES-OTR v3.1

The described attacks apply to OTR with parallel processing of AD. A brief description of OTR can be found in Appendix C.5.

***L*-recovery attack.** If we use the same nonce N 2^{64} times, we can recover L:

```
1: for $i \leftarrow 1$ to $2^{64}$ do query $C\|T \leftarrow \mathcal{E}_K(N, \varepsilon, M^i)$ with fresh $M^i \in \{0,1\}^{4\cdot 128}$.
2: Find $i \neq j$ s.t. $C_1^i \oplus M_2^i = C_3^j \oplus M_4^j$, compute $L = (M_1^i \oplus M_3^j) \cdot (1 \oplus 2)^{-1}$.
```

To avoid nonce reuse, we can encrypt a huge random message (with $|M|_{128} \approx 2^{64}$) with a nonce N and look for an internal collision with $i \neq j$:

$$C_{2i} \oplus M_{2i-1} = C_{2j} \oplus M_{2j-1} \text{ implying } C_{2i-1} \oplus 2^{i-1} \cdot 2 \cdot L = C_{2j-1} \oplus 2^{j-1} \cdot 2 \cdot L,$$

revealing L for this N. We further expect to find many values of $1 \leq i \leq |M|_{128}/2$ for which $2^{i-1} \cdot L \oplus M_{2i-1}$ (or $2^{i-1} \cdot 3 \cdot L \oplus C_{2i-1}$) will be a string of the form $\epsilon(\tau)\|1\|N'$. For any such N' we can use $L' = C_{2i-1}$ (or $L' = C_{2i}$) to bootstrap the following attack.

***E_K* oracle (using (N, L) pair).** Assuming that we know an (N, L) pair (either from the L-recovery attack or from a previous execution of the present E_K oracle), by a single encryption query with nonce N we can obtain $E_K(x_1), \ldots, E_K(x_r)$ for a list $x_1, \ldots, x_r$ as follows:

```
1: for $i \leftarrow 1$ to $r$ do set $M_{2i-1} \leftarrow x_i \oplus 2^{2i-1} \cdot L$ and pick $M_{2i}$ arbitrarily.
2: Query $C\|T \leftarrow \mathcal{E}_K(N, \varepsilon, M)$.
3: for $i \leftarrow 1$ to $r$ do compute $E_K(x_i) = M_{2i} \oplus C_{2i-1}$.
```

In each execution of this attack, we can add one block to the list of x_i-s to prepare a fresh pair N', L' for the next execution of the attack, allowing for its nonce respecting repetition. The oracle E_K can be used to mount universal forgeries and CPA decryption attacks with tiny complexity (but needs L).

9.11 CLOC

A description of CLOC can be found in Appendix C.6.

***E_K* oracle in CLOC (nonce-respecting, beyond birthday bound).** In CLOC, the processing of AD and nonce has the form $V = f_1(f_2(K, A) \oplus \mathsf{ozp}(\mathsf{param}\|N))$ where the function f_1 is easy to invert. To compute $E_K(x)$ for an $x \in \{0,1\}^{128}$, we pick fixed AD A and do:

```
1: for $i \leftarrow 1$ to $2^{64}$ do query $C^i\|T^i \leftarrow \mathcal{E}_K(N^i, A, M^i)$ with random $M^i \in \{0,1\}^{2\cdot 128}$.
2: Find $i \neq j$ s.t. $M_1^i \oplus C_1^i = M_2^j \oplus C_2^j$, compute $W \leftarrow f_1^{-1}(\mathsf{fix1}(C_1^j)) \oplus \mathsf{ozp}(\mathsf{param}\|N^i)$.
3: if $f_1^{-1}(x) \oplus W$ of the form $\mathsf{ozp}(\mathsf{param}\|\bar{N})$ query $E_K(x)\|T \leftarrow \mathcal{E}_K(\bar{N}, A, 0^{128})$.
4: else abort.
```

The attack works because the collision on line 2 implies that $V^i = \mathsf{fix1}(C_1^j)$ so we deduce the V^i value for a random nonce N^i with A. This allows us to recover $W = f_2(K, A)$.

If x is not of the correct form, it is bad luck. When using nonces of 112 bits, which is the maximum, the probability to have the correct form is 2^{-16}. But we can run this attack 2^{16} times to get many $W^i = f_2(K, A^i)$ with complexity 2^{80}. Then at least one W^i will be such that $f_1^{-1}(x) \oplus W_i$ is of the correct format for any x.

This attack does not work on SILC, in which W depends on both N and A.

Universal forgery and CPA decryption attack in CLOC (nonce-respecting, beyond birthday bound). With the previous E_K oracle, we can simulate the encryption or the decryption process and thus mount universal forgeries and CPA decryption.

9.12 Jambu

A brief description of Jambu is given in Appendix C.17.

Universal forgery (number of decryption queries at 64-bit-bound). Because the tags of Jambu are only 64-bit long, the trivial tag guessing attack only requires 2^{64} decryption queries. We can make a forgery for any target triplet (N, A, M) (with $|M|_{64} = m$) with a single encryption query, but requiring many decryption queries:

```
1: Pick $M'_m \in \{0,1\}^{|M_m|}$ s.t. $M_m \neq M'_m$, set $M' \leftarrow M_1 \| \ldots \| M_{m-1} \| M'_m$
2: Query $C' \| T' \leftarrow \mathcal{E}_K(N, A, M')$, set $C_m \leftarrow C'_m \oplus M_m \oplus M'_m$
3: for $T \in \{0,1\}^{64}$ do try forging with $(N, A, C'_1 \| \ldots \| C'_{m-1} \| C_m \| T)$
```

Semi-universal forgery (beyond 64-bit-bound, nonce-misusing). As Jambu uses a state of 192 bits, collision-based attacks always require a data complexity of about 2^{96} queries. We can therefore forge a ciphertext for N, M with about 2^{96} encryption queries, but we cannot choose the AD and the attack is nonce-misusing. We pick $M' \in \{0,1\}^{128}$ and do:

```
1: for $i \leftarrow 1$ to $2^{96}$ do query $C^i \| T^i \leftarrow \mathcal{E}_K(N, A^i, M')$ with distinct $A^i \in \{0,1\}^{127}$
2: Find $i \neq j$ s.t. $C^i, T^i = C^j, T^j$
3: Query $C \| T \leftarrow \mathcal{E}_K(N, A^i, M)$, forge with $(N, A^j, C \| T)$
```

In the two queries with colliding ciphertexts the internal states just after the AD processing must collide, allowing the forgery. A natural question that arises is whether such attack cannot be done in a nonce respecting way, using single-block ADs. The answer is no, because we need about 2^{96} queries to find the internal collision with good probability, but there are only 2^{64} values of the nonce in Jambu, so this approach is impossible.

Existential forgery (beyond 64-bit-bound, nonce-respecting). The problem of having too few nonces in the previous attack can be circumvented if we fold many subqueries into every encryption query. We pick a 64-bit constant string P. We then do:

```
1: for $i \leftarrow 1$ to $2^{48}$ do query $C^i \| T^i \leftarrow \mathcal{E}_K(N^i, A^i, M^i)$ with $M^i = \|_{j=1}^{2^{48}} Q^i_j \| P \| P$
2: Find $(i, j) \neq (i', j')$ such that $j \equiv j' \equiv 2 \pmod 3$ and $C^i_j \| C^i_{j+1} = C^{i'}_{j'} \| C^{i'}_{j'}$
3: Forge with $(N^i, A^i, C^i_1 \| \ldots \| C^i_{j-1} \| C^{i'}_{j'} \| \ldots \| C^{i'}_{2^{48}} \| T^{i'})$
```

The collision on line 2 implies that the i^{th} and i'^{th} queries had an internal state-collision $R^i_j \| U^i_j \| V^i_j = R^{i'}_{j'} \| U^{i'}_{j'} \| V^{i'}_{j'}$ on the j^{th} and j'^{th} state respectively with overwhelming probability. Because of this, the forgery succeeds. We managed to forge with only 2^{48}

encryption queries, but we still have to process about 2^{96} blocks of data in those queries. A big drawback if this strategy is that it cannot be used for a universal forgery.

9.13 Tiaoxin-346

We give a brief description of Tiaoxin in Appendix C.8. Note that we change the meaning of subscript and square brackets compared to the original Tiaoxin description [Nik16], i.e. $T[j]$ denotes a vector of j 128-bit strings and $T[j]_i$ with $i \leq j$ denotes i^{th} block of the vector $T[j]$.

Nonce-misuse key recovery. We pick $M, \bar{M}, \tilde{M} \in \{0,1\}^{4\cdot 128}$ such that $M_i \oplus \bar{M}_i = \Delta$ and $M_i \oplus \tilde{M}_i = \tilde{\Delta}$ for $i = 0,1,2,3$ with $\Delta \neq \tilde{\Delta}$. We pick arbitrary N and A and recover two 128 bit words $T'[4]_0$ and $T'[3]_0$ of the internal state right after processing of N, A and the first two blocks of M by:

```
1: Query $C\|T \leftarrow \mathcal{E}_K(N, A, M)$, $\bar{C}\|\bar{T} \leftarrow \mathcal{E}_K(N, A, \bar{M})$ and $\tilde{C}\|\tilde{T} \leftarrow \mathcal{E}_K(N, A, \tilde{M})$.
2: for $i \leftarrow 2, 3$ do set $\gamma_i \leftarrow \mathsf{ShiftRows}^{-1}(\mathsf{MixColumns}^{-1}(\bar{C}_i \oplus C_i))$.
3: for $i \leftarrow 2, 3$ do set $\tilde{\gamma}_i \leftarrow \mathsf{ShiftRows}^{-1}(\mathsf{MixColumns}^{-1}(\tilde{C}_i \oplus C_i))$.
4: for byte index $j \leftarrow 0$ to 15 do
5:     for $i \leftarrow 2, 3$ do Find set of solutions $X_{i,j}$ of $\gamma_{i,j} = \mathsf{SubBytes}(x) \oplus \mathsf{SubBytes}(x \oplus \Delta)$.
6:     for $i \leftarrow 2, 3$ do Find set of solutions $\tilde{X}_{i,j}$ of $\tilde{\gamma}_{i,j} = \mathsf{SubBytes}(x) \oplus \mathsf{SubBytes}(x \oplus \tilde{\Delta})$.
7:     Set $T'[4]_{0,j} \leftarrow X_{2,j} \cap \tilde{X}_{2,j}$ and $T'[3]_{0,j} \leftarrow X_{3,j} \cap \tilde{X}_{3,j}$.
8: end for
```

The above works, as we can verify that in the encryption of M we have

1. $T'[3] = R(T[3], M_0)$,
2. $T'[4] = R(T[4], M_1)$,
3. $T'[6] = R(T[6], M_0 \oplus M_1)$,
4. $C_0 = T'[3]_0 \oplus T'[3]_2 \oplus T'[4]_1 \oplus (T'[6]_3 \& T'[4]_3)$,
5. $C_1 = T'[6]_0 \oplus T'[4]_2 \oplus T'[3]_1 \oplus (T'[6]_5 \& T'[3]_2)$,
6. $T''[3] = R(T'[3], M_2)$,
7. $T''[4] = R(T'[4], M_3)$,
8. $T''[6] = R(T'[6], M_2 \oplus M_3)$,
9. $C_2 = T''[3]_0 \oplus T''[3]_2 \oplus T''[4]_1 \oplus (T''[6]_3 \& T''[4]_3)$,
10. $C_3 = T''[6]_0 \oplus T''[4]_2 \oplus T''[3]_1 \oplus (T''[6]_5 \& T''[3]_2)$.

In the encryption of $\bar{M}$ we have the following (and similar for $\tilde{M}$ and $\tilde{\Delta}$)

1. $\bar{T}'[3] = R(T[3], M_0 \oplus \Delta)$,
2. $\bar{T}'[4] = R(T[4], M_1 \oplus \Delta)$,
3. $T'[6] = R(T[6], M_0 \oplus M_1)$,
4. $\bar{C}_0 = \bar{T}'[3]_0 \oplus \bar{T}'[3]_2 \oplus \bar{T}'[4]_1 \oplus (T'[6]_3 \& \bar{T}'[4]_3)$,
5. $\bar{C}_1 = T'[6]_0 \oplus \bar{T}'[4]_2 \oplus \bar{T}'[3]_{10} \oplus (T'[6]_5 \& \bar{T}'[3]_2)$,
6. $\bar{T}''[3] = R(\bar{T}'[3], M_2 \oplus \Delta)$,
7. $\bar{T}''[4] = R(\bar{T}'[4], M_3 \oplus \Delta)$,
8. $T''[6] = R(T'[6], M_2 \oplus M_3)$,
9. $\bar{C}_2 = \bar{T}''[3]_0 \oplus \bar{T}''[3]_2 \oplus \bar{T}''[4]_1 \oplus (T''[6]_3 \& \bar{T}''[4]_3)$,
10. $\bar{C}_3 = T''[6]_0 \oplus \bar{T}''[4]_2 \oplus \bar{T}''[3]_1 \oplus (T''[6]_5 \& \bar{T}''[3]_2)$.

We can easily see that

$$\bar{T}'[3] \oplus T'[3] = (\Delta, 0, 0) \text{ and } \bar{T}''[3] \oplus T''[3] = (0, A(T'[3]_0) \oplus A(T'[3]_0 \oplus \Delta), 0),$$

and that

$$\bar{T}'[4] \oplus T'[4] = (\Delta, 0, 0, 0) \text{ and } \bar{T}''[4] \oplus T''[4] = (0, A(T'[4]_0) \oplus A(T'[4]_0 \oplus \Delta), 0, 0).$$

It follows that the differences of ciphertext blocks used in the lines 5 and 6 are a result of a differential equation for a single round of AES. This can be reduced to a collection of 16 differential equations for AES Sbox, allowing to recover the parts of the secret state as intersections of solutions found in the said lines (we can check that we always have $|S_{i,j} \cap \tilde{S}_{i,j}| = 1$).

We can then repeat this process with longer messages to obtain $T[3]$ and $T[4]$ and we recover $T'[4]$ and $T'[3]$ with 12 queries (3 queries per 128-bit word of $T[4]$). The state $T[6]$ follows in a similar method using 18 queries. Once the state $(T[3], T[4], T[6])$ is recovered, we invert the initialization and obtain K.

9.14 AEGIS v1.1

A brief description of AEGIS is given in Appendix C.9.

Universal forgery and decryption attack (tiny complexity, nonce-misuse). To forge for (N, A, M) or to decrypt (N, A, C, T), we only need to recover the secret state S after processing A with nonce N, the rest of encryption/decryption can then be reconstructed.

We pick three messages $M', \bar{M}, \tilde{M} \in \{0,1\}^{3 \cdot 128}$ with the same criteria as for Tiaoxin (with $\Delta \neq \tilde{\Delta}$). To recover S'_0, a part of the state S' right after processing M'_1 with N and A, we:

```
1: Query $C'\|T' \leftarrow \mathcal{E}_K(N, A, M')$, $\bar{C}\|\bar{T} \leftarrow \mathcal{E}_K(N, A, \bar{M})$ and $\tilde{C}\|\tilde{T} \leftarrow \mathcal{E}_K(N, A, \tilde{M})$.
2: Set $\gamma \leftarrow \mathsf{ShiftRows}^{-1}(\mathsf{MixColumns}^{-1}(\bar{C}_3 \oplus \bar{M}_3 \oplus C'_3 \oplus M'_3))$.
3: Set $\tilde{\gamma} \leftarrow \mathsf{ShiftRows}^{-1}(\mathsf{MixColumns}^{-1}(\tilde{C}_3 \oplus \tilde{M}_3 \oplus C'_3 \oplus M'_3))$.
4: Recover bytes of $S'_0$ using $\gamma, \tilde{\gamma}, \Delta, \tilde{\Delta}$ in differential equations as with Tiaoxin.
```

The attack works because the difference $(C'_3 \oplus M'_3) \oplus (\bar{C}_3 \oplus \bar{M}_3)$ (associated to $M'_1 \neq \bar{M}_1$) is equal to the difference $R(R(S_4) \oplus S_0 \oplus M'_1) \oplus R(R(S_4) \oplus S_0 \oplus \bar{M}_1)$ (where $R(S_4) \oplus S_0 = S'_0$), with R just a single AES round. We can repeat this strategy to recover the remaining four 128-bit words of $S'_1, \ldots, S'_4$ with 3 queries each. Then we can recover S, having done 15 nonce reusing queries. The possibility of a low-complexity nonce reusing attack is mentioned in the AEGIS v1.1 specifications [WP16].

9.15 ACORN v3

A brief description of ACORN is given in Appendix C.10.

Universal forgery and decryption attack (tiny complexity, nonce-misuse). To forge the encryption of (N, A, M) or to decrypt (N, A, C, T), we only need to recover the internal state S_o after processing N, A, which allows to finish the rest of encryption/decryption. We sketch the main idea of the attack.

We make two encryption queries $C^1\|T^1 \leftarrow \mathcal{E}_K(N, A, 0\|B)$ and $C^2\|T^2 \leftarrow \mathcal{E}_K(N, A, 1\|B)$ for any $B \in \{0,1\}^{58}$. We can see that ks^j_{i+o} is constant for $j = 1, 2$ and $i = 0, \ldots, 57$ and that $\mathsf{ks}^1_{58+o} \oplus \mathsf{ks}^2_{58+o} = S_{58+o,61} \oplus S_{58+o,193}$, which is a linear equation in the bits of S_o. We recover 292 more equations by making 292 pairs of (longer) queries that differ only in a single bit, and solve the system for S_o. The knowledge of S_o allows arbitrary forgeries and decryptions with N, A.

9.16 Ketje

Key recovery (tiny complexity, nonce-misusing). The authors of Ketje themselves point at the possibility of this attack. Because Ketje uses only a single round of the Keccak$-f$ function [BDP+16a], the diffusion between two consecutive sponge states is low. In addition, the algebraic degree of a single round of Keccak$-f$ is only 2. We use this to recover the internal state S after processing of N and A, and then the secret key K by inverting the processing of N, A. We sketch the main idea of the attack.

We make queries $C^i\|T^i \leftarrow \mathcal{E}_K(N, A, M^i)$ with some fixed (N, A) and $M^i \in \{0,1\}^{2\cdot(r-4)}$ s.t. $M^i_2 = 0^r$ for $i = 1, \ldots, \theta$. For each i we can use M^i_1 and C^i_2 to derive degree-2 polynomial equations with the bits in the inner (capacity) part of S (marked red in Figure C.1) as unknowns. Each bit in C^i_2 depends on 31 bits of the previous state on average [BDPVA09], so we expect an overwhelming majority of the bits of the attacked state to be covered by the derived equations. We need the number of nonce misusing queries θ to be a small multiple of $\frac{b-r+4}{r-4} = 11{,}5$ in order to fully determine the system. Moreover, no more than a single unique monomial of degree 2 per every bit of the state appears in the system, so with $\theta = 60$, we should be able to linearize and solve the system for S.

9.17 Morus

We give a brief description of Morus 640 in Appendix C.12.

Nonce-misuse universal forgery and CPA decryption. If we recover the state S right after the initialization with N, we can forge ciphertexts with this N and decrypt any ciphertext using this N. We sketch the S recovery attack.

We first recover S_2 and S_3 by querying $C^i\|T^i \leftarrow \mathcal{E}_K(N, \varepsilon, M^i)$ with $M^i \in \{0,1\}^{256}$ for $i = 1, \ldots, 4$. Letting $\delta_i = M^1_0 \oplus M^i_0$ with $i \neq 1$, we have that

$$\begin{aligned}(C^1 \oplus M^1) \oplus (C^i \oplus M^i) =& (\mathsf{Rotl}(\delta_i, b_1) \lll (w_3 + 96)) \oplus S_2 \& \mathsf{Rotl}(\delta_i \oplus \mathsf{Rotl}(\delta_i, b_1), b_3) \\ & \oplus S_3 \& (\mathsf{Rotl}(\delta_i, b_2) \lll w_4) \\ & \oplus (\mathsf{Rotl}(\delta_i, b_2) \lll w_4) \& \mathsf{Rotl}(\delta_i \oplus \mathsf{Rotl}(\delta_i, b_1), b_3),\end{aligned}$$

where Rotl is a linear function, $\lll$ denotes a circular rotation, and all b_r-s and w_t-s are constants. Each δ_i provides 128 linear equations in 256 binary unknowns, so with $\delta_1, \delta_2, \delta_3$, we are able to recover the values of S_2 and S_3 with high probability. Once S_2 and S_3 are known, $C_1^1 \oplus M_1^1$ can be expressed as a linear function of S_0 and S_1 and we learn their xor-difference.

We still need to recover S_0, S_1, S_4, i.e. 384 bits, and have 128 linear equations (so 256 unknown bits). We query $\bar{C}^j \| \bar{T}^j \leftarrow \mathcal{E}_K(N, \varepsilon, \bar{M}^j)$ with $\bar{M}^j = M_0^1 \| \bar{M}_1^j \| 0^{128}$ and $\bar{M}_1^j \in \{0,1\}^{128}$ for $j = 1, \ldots, \theta$. Each $\bar{C}_2^j$ will supply 128 polynomial equations in S_0, S_1, S_4 of degree at most 3. By examining the StateUpdate and the keystream generation functions of Morus, we verify that there will be no more than $19 \cdot 128$ unique monomials of degree higher than 1 present in all equations in the worst case and only $9.25 \cdot 128$ on average. Thus by taking $\theta = 16$, we should be able to linearise the system and recover S_0, S_1 and S_4 with high probability, using 20 queries for the entire attack.

9.18 COLM

We briefly describe COLM in Appendix C.16.

Below, we first observe that an L-recovery attack would allow to easily make universal forgeries and CCA decryption attacks. Next, we will see two methods to extract L. One is nonce-respecting with complexity beyond the birthday bound. The other has nonce-misuse, at the 64-bit-bound complexity. We conclude with an easy nonce-respecting existential forgery attack as the birthday bound complexity.

CCA decryption attack (tiny complexity, using L). To decrypt (N, A, C) where $|A|_{128} \geq 2$ we do:

```
1: Set $A'_1 = A_2 + 3 \cdot 6 \cdot L$ and $A'_2 = A_1 + 3 \cdot 6 \cdot L$ and $A'_i = A_i$ for $i = 3, \ldots, |A|_{128}$
2: Query $M \leftarrow \mathcal{D}_K(N, A', C)$
3: return $M$
```

The attack works with a single decryption query because $\mathsf{AA}'_1 = A'_1 + 3 \cdot 2 \cdot L = A_2 + 3 \cdot 4 \cdot L = AA_2$ and $\mathsf{AA}'_2 = A'_2 + 3 \cdot 4 \cdot L = A_1 + 3 \cdot 2 \cdot L = AA_1$, so A' produces the same IV as A.

Universal forgery (tiny complexity, using L). To make a forgery, we construct an E_K oracle in a CPA attack (using the knowledge of L). This allows us to simulate the encryption process and create a valid ciphertext for arbitrary (N, A, M). To compute $E_K(x)$, we start with arbitrary $\bar{N} \neq N$ and do:

```
1: Set $\bar{A} = \bar{N} \| \mathsf{param} \oplus 3^2 \cdot L$.
2: for $i \leftarrow 1$ to 135 do set $\bar{M}_i \leftarrow 2^i \cdot L$ and $\bar{Y}_i \leftarrow 2^{i-1} \cdot L$.
3: Query $\bar{C} \leftarrow \mathcal{E}_K(\bar{N}, \bar{A}, \bar{M})$.
4: for $i \leftarrow 1$ to 135 do set $\bar{\mathsf{CC}}_i \leftarrow \bar{C}_i \oplus 2^i \cdot L$.
5: Find $\{(x_i, y_i)\}_{i=1}^{128} \subset \{(\bar{Y}_j, \bar{\mathsf{CC}}_j)\}_{j=1}^{135}$ with $y_i$-s linearly independent.
```

```
6: Find $S = \{s_1, \ldots, s_{|S|}\} \subseteq \{1, \ldots, 128\}$ s.t. $x = 3 \cdot \bigoplus_{i \in S} y_i \oplus L$.
7: Pick an $N$, set $A_1 \leftarrow N \| \mathsf{param} \oplus 3^2 \cdot L$ and $M \leftarrow 2 \cdot L$.
8: for $i \leftarrow 1$ to $|S|$ do $A_{i+1} \leftarrow x_{s_i} \oplus 3 \cdot 2^{i+1} \cdot L$.
9: Query $C \leftarrow \mathcal{E}_K(N, A, M)$, compute $E_K(x) \leftarrow C_1 \oplus 2 \cdot L$.
```

In the query made in line 3, $\bar{A}$ is constructed to make sure that $\bar{\mathsf{AA}}_1 = \bar{\mathsf{AA}}_0$, forcing $\bar{\mathsf{IV}} = 0$, and M makes sure that MM consists of $135 = 128 + \log_2(128)$ zero blocks. Hence $\bar{X}_i = L$ and $\bar{Y}_i = L + 3 \cdot (2^{i-1} - 1) \cdot L = 2^{i-1} \cdot L$ for $i = 1, \ldots, 135$. Each $\bar{Y}_i$ will be distinct (because 2 is a primitive root of $\mathsf{GF}(2^{128})$ in COLM) and $\bar{\mathsf{CC}}_i = E_K(\bar{Y}_i)$. We find 128 linearly independent values $y_1, \ldots, y_{128}$ among $\bar{\mathsf{CC}}_1, \ldots, \bar{\mathsf{CC}}_{135}$ with probability close to 1.

With the obtained list, we can find the set S (in line 6) for an arbitrary x. In the query made in line 9, we force $\mathsf{IV} = \sum_{i \in S} y_i$. and $\mathsf{MM}_1 = 0$, and $Y_1 = X_1 + 3 \cdot \mathsf{IV} = x$. The total query complexity is small: there is one encryption query for precomputation, and then, to forge an encryption, we need one query per E_K evaluation, so $1 + a + 2(\ell + 1)$ encryption queries. This is negligible compared to the complexity of getting L.

***L*-recovery attack (low success probability).** This attack must guess the last 64 bits of $3^2 \cdot L$ because param is a constant for a fixed instance of COLM. We pick a constant message $M \in \{0,1\}^{128}$ and a random block $B\{0,1\}^{128}$ (the last 64 bits of B are our guess). With distinct $N^1, \ldots, N^{2^{32}}$ we do:

```
1: for $i \leftarrow 1$ to $2^{32}$ do query $C^i \leftarrow \mathcal{E}_K(N^i, A^i, M)$ with $A^i = (N^i \| \mathsf{param}) \oplus B$.
2: Find $i \neq j$ s.t. $C^i = C^j$, compute $L = ((N^i \| \mathsf{param}) \oplus (N^j, \mathsf{param}) \oplus B) \cdot (3^2)^{-1}$.
```

Clearly, $\mathsf{AA}_0 + \mathsf{AA}_1 = B + 3^2 \cdot L$. If the guess B is correct, AA_0, AA_1, and $3 \cdot L$ always end with the same 64 bits. Due to the birthday attack (on the remaining bits), we must have $i \neq j$ s.t. $\mathsf{AA}_0^i = \mathsf{AA}_1^j$, so $(N^i, \mathsf{param}) + 3 \cdot L = \mathsf{AA}_0^i = \mathsf{AA}_1^j = (N^j \| \mathsf{param}) + B + 3 \cdot 2 \cdot L$ and thus $\mathsf{AA}_0^j = (N^j \| \mathsf{param}) + 3 \cdot L = (N^i, \mathsf{param}) + B + 3 \cdot 2 \cdot L = \mathsf{AA}_1^i$. This means that N^i, A^i and N^j, A^j produce the same IV, and this collision on IV induces a collision on C. The total complexity is 2^{32} encryptions (with messages of one block and A of 1 block).

We note that the attack is nonce-respecting, although the probability of success is only 2^{-64}. This can be improved if we assume *parameter* misuse, i.e. existence of several instances using the same key with different parameters.

***L*-recovery attack (nonce-misuse, beyond birthday bound).** We mount a similar attack, assuming that every nonce can be repeated 2^m times. We use 2^{128-2m} distinct nonce $N^1, \ldots, N^{2^{128-2m}}$, AD of two blocks and a random block $B \in \{0,1\}^{128}$ to do:

```
1: for $i \leftarrow 1$ to $2^{128-2m}$ do
2:     for $j \leftarrow 1$ to $2^m$ do query $C^{i,j} \leftarrow \mathcal{E}_K(N^i, A^{i,j}, M)$ s.t. $A_2^{i,j} = A_1^{i,j} \oplus B$.
3: end for
4: Find $i, j \neq j'$ s.t. $C^{i,j} = C^{i,j'}$, compute $L \leftarrow (A_1^{i,j} \oplus A_1^{i,j'} \oplus B) \cdot (3 \cdot 6)^{-1}$.
```

We note that when $m = 64$, this attack has birthday complexity.

9.19 Deoxys v1.41

A brief description of Deoxys-II is given in Appendix C.7.

Semi-universal forgery and CCA decryption attack on Deoxys-II (reusable, nonce-misuse). The encryption algorithm of Deoxys-II has the form $\mathcal{E}_K(N, A, M) = \bar{\mathcal{E}}(K, N, f_2(f_1(K, A), M), M)$ where $\bar{\mathcal{E}}$ produces a (stretched) ciphertext and f_1 and f_2 are keyed functions with constant-size output. The attack is based on finding a collision on f_1. Assuming each nonce can be used up to 2^m times, to forge for (N, M) we use $N^1, \ldots, N^{2^{128-2m}} \neq N$ all distinct and $M' \neq M$ of 2 blocks, and do:

```
1: for $i \leftarrow 1$ to $2^{128-2m}$ do
2:     for $j \leftarrow 1$ to $2^m$ do query $C^{i,j} \| T^{i,j} \leftarrow \mathcal{E}_K(N^i, A^{i,j}, M')$ with random $A^{i,j}$.
3: end for
4: Find $i, j \neq j'$ s.t. $A^{i,j} \neq A^{i,j'}$ and $T^{i,j} = T^{i,j'}$.
5: Query $C \| T \leftarrow \mathcal{E}_K(N, A^{i,j}, M)$ and forge with $(N, A^{i,j'}, C \| T)$.
```

We can modify this attack to decrypt $(N, A^{i,j}, C\|T)$ by making a CCA decryption query on $(N, A^{i,j'}, C, T)$. This can only decrypt messages using $A^{i,j}$ as associated data. The total complexity of the attack is 2^{128-m} queries. Note that if $m = 64$, the complexity becomes birthday bounded.

Chapter 10

Conclusion and Future Work

In this thesis, we present a rather diverse collection of results from the field of AE. Although these results are mostly unrelated to one another, they all spring from the same motivation: to push the state of the art towards *efficient*, *provably secure* authenticated encryption that is *relevant for practical applications.*

This is a not-so-concrete aspirational goal, and there are many milestones along the way to achieving it.[1] We contribute our bit to several paths that lead to our goal.

Design of AE schemes. The most obvious way of converging to our goal is the design of efficient, provably secure schemes. While the design of such schemes appears in every chapter (except Chapter 9), Chapters 3 to 6 are specifically dedicated to design of provably secure AE schemes.

We present OMD, a "basic" nonce-based AE scheme, in Chapter 3. The main pros of (the instances of) OMD, as the author sees them, are its contribution to cryptographic diversity, its low overhead beyond the compression function calls, and the high quantitative security level combined with decent performance.

The misuse-resistant variants of OMD inherit most of the enumerated properties, and add nonce-reuse resistance. The most interesting component of MR-OMD (and PMR-OMD) is perhaps the highly efficient, parallelizable PRF dedicated to the domain of a nonce-based AE scheme.

The work on p-OMD was inspired by the question whether it is possible to construct a secure nonce-based AE scheme *purely* based on the masked Merkle-Damgård iteration, but more efficiently than by the obvious sequential "first-AD-then-message" approach. We proved that the AE processing can be dealt with by "absorbing" the blocks of AD into the intermediate chaining values of the Merkle-Damgård iteration. This simple algorithmic modification brought a substantial boost in performance: in a typical situation, the cost of processing AD became almost free of charge.

[1]It would be more precise to say "converging to it" instead of "achieving it." The author believes that a necessary condition for advancing towards this ambitious goal is accepting that a perfect solution may not exist.

We quickly realized, that such a modification can also be applied to sponge-based AE schemes. Unlike others who investigated full-state absorption, we proved the security of the general variable output-length FKS and the FKD constructions, and showed how AE schemes can benefit from this by a *modular* security analysis. We also put a lot of effort in making the proof as concise and easily verifiable as possible, because this aspect of provable security is as important as the correctness of the proof itself.

Design of new notions. Another crucial component, some may even say the most important component,[2] of achieving the goal of secure and useful AE is the design and critical evaluation of security notions.

Our work on the nonce-misuse resistance of online authenticated encryption was induced by a sudden surge of popularity of the OAE1 notion around the start of CAESAR competition. Our concerns were about not-yet understood properties captured by OAE1, coupled with strong verbal claims of security that accompanied it. We attempted to dissect the tangle of claims that advertised the security and usefulness of OAE1 schemes from the actual constraints implied by an "online" setting. We presented two attacks that shed light on the information leakage of OAE1, and proposed a definition that better captures the intuition of *best possible security* in the *online setting.* It is noteworthy that the new notion of OAE2 is not some strengthened generalization of OAE1; it has a completely different syntax and philosophy, both of them attempting to capture what is really *relevant* in a setting, when onlineness of encryption would be needed.

The security notions we propose for variable stretch AE are also based on the struggle to capture what is relevant for practice. We argue that an extension of basic NAE security that tolerates variation of ciphertext expansion is of practical interest. We define a suite of notions, but most of them only serve the purpose of establishing relations, and as verification that we are on a right track. Only two notions are assumed to be "exposed" to a designer: NVAE and KESS, the latter being a tool for easier tweaking of existing schemes. The NVAE notion itself is unusual in that it is parameterized. Although a bound is always a function of some fixed stretch τ_c, there is an informal, but natural definition of general NVAE security.

Cryptanalysis. While the author considers the provable security approach indispensable, it should always be taken with a grain of salt. For instance, it is debatable whether a nonce-based AE scheme that is perfectly secure with non-repeating nonces, but succumbs to a low-complexity *key recovery* attack if nonces accidentally repeat, is fit for widespread adoption. The problem is that this information is invisible if one only looks at the security claims with respect to the NAE notion.

Given a relatively large number of schemes that make very similar security claims, this kind of information can be very useful if one needs to, say, pick only a few to form

[2]See a short commentary on the importance of provable security in paragraph "Controversy surrounding provable security."

a portfolio.[3] Thus motivated, we compile Table 9.2 that provides just such information.

Controversy surrounding provable security. The provable security treatment has both its supporters and critics. The critics, e.g. Koblitz and Menezes [KM04, KM06, KM11], claim that the provable security analysis is of limited, or even no use to the real world applications of cryptography. The criticism has been succinctly summarised by Damgård [Dam07]:

- "A proof of security never proves security in an absolute sense, it relates security to an unproven assumption that some computational problem is hard." [Dam07, p. 4]
- "The quality of a security reduction should not be ignored – it matters how tight it is, and how strong the underlying assumption is." [Dam07, p. 4]
- "A security reduction only proves something in a particular model specifying what the adversary has access to and can do." [Dam07, p. 4]

The author of this thesis agrees with Damgård, in that the security reductions are a useful tool as long as their *interpretation* is done carefully, and that precise formal definitions of security should *always* be a design goal of any cryptographic primitive. After all, how is one supposed to find a good solution to a problem if one does not know what problem is he/she *exactly* solving?

Future work. One of the reasons that made OMD finish in the 2nd round of CAESAR was probably its rather average performance. Yet, the measurements that were produced by the end of the 2nd round did not reflect OMD's full potential, as there was no hardware available to run OMD implemented with the Intel SHA Extensions. Nowadays, this instruction extension set is already deployed, and it would be interesting to see the exact improvement of OMD's performance. The same applies to p-OMD, which could become a very attractive AE scheme one implemented with Intel SHA Extension.

Anther improvement that could be done to p-OMD would be its simplification. We note that results already exist in this direction, as such a simplification has already been proposed by Ashur and Mennink [AM16].

There is another possible research avenue suggested to the author by Peter Gaži, namely to check whether the security bounds of OMD are tight, and possibly perform a dedicated non-modular analysis to improve the bounds.

As mentioned in Chapter 6, our results for FKS are done in the ideal-permutation model. Revisiting the analysis of FKS and FKD in the standard model, e.g. using the framework of Soni and Tessaro [ST17], would be of interest.

[3] A small caveat occurs if *all* candidates that are very fast also succumb to nonce-misusing key recoveries.

The OAE2-secure construction **CHAIN** is based on a modular design, that uses a nonce-based AE scheme as a blackbox subroutine. It may be interesting to design a dedicated construction, based on e.g. tweakable blockciphers.

Finally, it would also be interesting to extend the security notions for variable-stretch AE to additionally capture security against re-forgeries for the target stretch τ_c.

List of Definitions

List of Figures

Bibliography

[AA14] Ralph Ankele and Robin Ankele. Fast software implementation of Offset Merkle-Damgård (OMD). Student project at Ecole Polytechnique Fédéral Lausanne, 2014. *Cited on pages:* 33, 50, 53, 56, 253, 254, 255, 256, 257, and 258.

[AA16] Ralph Ankele and Robin Ankele. Software benchmarking of the 2^{nd} round CAESAR candidates. *IACR Cryptology ePrint Archive*, 2016:740, 2016. *Cited on pages:* 55 and 56.

[AAB14] Javad Alizadeh, Mohammad Reza Aref, and Nasour Bagheri. Artemia v1, March 2014. *Cited on pages:* 96 and 260.

[ABB+14a] Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Florian Mendel, Bart Mennink, Nicky Mouha, Qingju Wang, and Kan Yasuda. PRIMATEs v1, March 2014. *Cited on page:* 96.

[ABB+14b] Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Florian Mendel, Bart Mennink, Nicky Mouha, Qingju Wang, and Kan Yasuda. PRIMATEs v1.02, September 2014. *Cited on page:* 9.

[ABB+14c] Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. APE: authenticated permutation-based encryption for lightweight cryptography. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*, pages 168–186. Springer, 2014. *Cited on page:* 260.

[ABD+16] Elena Andreeva, Andrey Bogdanov, Nilanjan Datta, Atul Luykx, Bart Mennink, Mridul Nandi, Elmar Tischhauser, and Kan Yasuda. COLM v1. `https://competitions.cr.yp.to/round3/colmv1.pdf`, 2016. *Cited on pages:* 201 and 272.

[ABL$^+$13] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Elmar Tischhauser, and Kan Yasuda. Parallelizable and authenticated online ciphers. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I*, volume 8269 of *Lecture Notes in Computer Science*, pages 424–443. Springer, 2013. *Cited on page:* 259.

[ABL$^+$14a] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to securely release unverified plaintext in authenticated encryption. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 105–125. Springer, 2014. *Cited on pages:* 5, 133, 163, 164, and 260.

[ABL$^+$14b] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Elmar Tischhauser, and Kan Yasuda. AES-COBRA v1, March 2014. *Cited on pages:* 260 and 261.

[ABL$^+$14c] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Elmar Tischhauser, and Kan Yasuda. AES-COPA v.1, March 2014. *Cited on pages:* 260 and 261.

[AC14] Mridul Nandi Avik Chakraborti. TriviA-ck-v1. `https://competitions.cr.yp.to/round1/triviackv1.pdf`, 2014. *Cited on page:* 260.

[ADMA15] Elena Andreeva, Joan Daemen, Bart Mennink, and Gilles Van Assche. Security of keyed sponge constructions using a modular proof approach. In Gregor Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 364–384. Springer, 2015. *Cited on pages:* 96, 102, 106, 108, and 114.

[AFF$^+$14a] Farzaneh Abed, Scott Fluhrer, John Foley, Christian Forler, Eik List, Stefan Lucks, David McGrew, and Jakob Wenzel. The POET Family of On-Line Authenticated Encryption Schemes Submission to the CAESAR competition, March 2014. *Cited on page:* 260.

[AFF$^+$14b] Farzaneh Abed, Scott R. Fluhrer, Christian Forler, Eik List, Stefan Lucks, David A. McGrew, and Jakob Wenzel. Pipelineable on-line encryption. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*, pages 205–223. Springer, 2014. *Cited on page:* 260.

[AFL16] Farzaneh Abed, Christian Forler, and Stefan Lucks. General classification of the authenticated encryption schemes for the CAESAR competition. *Computer Science Review*, 22:13–26, 2016. *Cited on pages:* 96 and 260.

[AJN14] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX v1, March 2014. *Cited on pages:* 96 and 260.

[AJN16] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX v3.0. `https://competitions.cr.yp.to/round3/norxv30.pdf`, 2016. *Cited on pages:* 201, 225, 270, and 271.

[AKL$^+$14] Farzaneh Abed, Stefan Kölbl, Martin M. Lauridsen, Christian Rechberger, and Tyge Tiessen. Authenticated encryption zoo. `https://aezoo.compute.dtu.dk`, 2014. *Cited on pages:* 260 and 261.

[ALMY14] Elena Andreeva, Atul Luykx, Bart Mennink, and Kan Yasuda. COBRA: A parallelizable authenticated online cipher without block cipher inverse. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*, pages 187–204. Springer, 2014. *Cited on pages:* 260 and 262.

[AM16] Tomer Ashur and Bart Mennink. Damaging, simplifying, and salvaging p-omd. In Matt Bishop and Anderson C. A. Nascimento, editors, *Information Security - 19th International Conference, ISC 2016, Honolulu, HI, USA, September 3-6, 2016, Proceedings*, volume 9866 of *Lecture Notes in Computer Science*, pages 73–92. Springer, 2016. *Cited on pages:* 75, 93, and 219.

[Bau01] Friedrich L. Bauer. Claude elwood shannon 1916–2001. *Informatik-Spektrum*, 24(4):228–229, Aug 2001. *Cited on page:* 1.

[BBKN01] Mihir Bellare, Alexandra Boldyreva, Lars R. Knudsen, and Chanathip Namprempre. Online ciphers and the hash-cbc construction. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 292–309. Springer, 2001. *Cited on pages:* 132 and 134.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1996. *Cited on page:* 3.

[BDJR97] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 394–403. IEEE Computer Society, 1997. *Cited on pages:* 2, 3, and 15.

[BDP$^+$14a] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. CAESAR submission: Ketje v1, March 2014. *Cited on page:* 96.

[BDP$^+$14b] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. CAESAR submission: Keyak v1, March 2014. *Cited on pages:* 96 and 260.

[BDP$^+$16a] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. CAESAR submisssion: Ketje v2. `https://competitions.cr.yp.to/round3/ketjev2.pdf`, 2016. *Cited on pages:* 196, 201, 212, 225, 269, and 270.

[BDP$^+$16b] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. CAESAR submisssion: Keyak v2. `https://competitions.cr.yp.to/round3/keyakv22.pdf`, 2016. *Cited on pages:* 96, 201, 225, 271, and 272.

[BDPA08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008. *Cited on pages:* 96 and 106.

[BDPA10] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge-based pseudo-random number generators. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2010. *Cited on page:* 95.

[BDPA11a] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, volume 7118 of *Lecture Notes in Computer Science*, pages 320–337. Springer, 2011. *Cited on pages:* 96, 98, 100, 113, 114, 117, 118, 133, 156, and 162.

[BDPA11b] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: single-pass authenticated encryption and other applications. *IACR Cryptology ePrint Archive*, 2011:499, 2011. *Cited on pages:* 100, 113, 117, and 118.

[BDPV07] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge Functions. ECRYPT Hash Workshop 2007, 2007. `http://csrc.nist.gov/groups/ST/hash/documents/JoanDaemen.pdf`. *Cited on pages:* 95 and 97.

[BDPV08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak Specifications. NIST SHA-3 Submission, 2008. `http://keccak.noekeon.org/`. *Cited on page:* 95.

[BDPV11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the Security of the Keyed Sponge Construction. In *Symmetric Key Encryption Workshop 2011*, 2011. *Cited on pages:* 95 and 96.

[BDPV12] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Permutation-Based Encryption, Authentication and Authenticated Encryption. In *Workshop Records of DIAC 2012*, pages 159–170, 2012. *Cited on pages:* 95, 96, and 162.

[BDPVA09] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak sponge function family main document. *Submission to NIST (Round 2)*, 3:30, 2009. *Cited on page:* 212.

[BEK16] Asli Bay, Oguzhan Ersoy, and Ferhat Karakoç. Universal forgery and key recovery attacks on elmd authenticated encryption algorithm. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 354–368, 2016. *Cited on page:* 196.

[Bel06a] Mihir Bellare. New proofs for NMAC and HMAC: security without collision-resistance. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 602–619. Springer, 2006. *Cited on page:* 34.

[Bel06b] Mihir Bellare. New proofs for NMAC and HMAC: security without collision-resistance. *IACR Cryptology ePrint Archive*, 2006:43, 2006. *Cited on page:* 35.

[Ber08] Daniel J Bernstein. Chacha, a variant of salsa20. In *Workshop Record of SASC*, volume 8, pages 3–5, 2008. *Cited on page:* 2.

[Ber14a] D. J. Bernstein. Cryptographic competitions: CAESAR. "`https://competitions.cr.yp.to/caesar-call.html`", 2014. *Cited on pages:* 4, 96, 165, and 198.

[Ber14b] D. J. Bernstein. Cryptographic competitions: Disasters. `https://competitions.cr.yp.to/disasters.html`, 2014. *Cited on pages:* 4 and 165.

[Ber16] Daniel J Bernstein. CAESAR use cases. `crypto-competitions` mailing list. July 16, 2016., 2016. *Cited on page:* 196.

[BGM04] Mihir Bellare, Oded Goldreich, and Anton Mityagin. The power of verification queries in message authentication and authenticated encryption. *IACR Cryptology ePrint Archive*, 2004:309, 2004. *Cited on pages:* 43, 47, 85, 89, and 127.

[BGR95] Mihir Bellare, Roch Guérin, and Phillip Rogaway. XOR macs: New methods for message authentication using finite pseudorandom functions. In Don Coppersmith, editor, *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, volume 963 of *Lecture Notes in Computer Science*, pages 15–28. Springer, 1995. *Cited on page:* 76.

[BGW01] Nikita Borisov, Ian Goldberg, and David A. Wagner. Intercepting mobile communications: the insecurity of 802.11. In Christopher Rose, editor, *MOBICOM 2001, Proceedings of the seventh annual international conference on Mobile computing and networking, Rome, Italy, July 16-21, 2001.*, pages 180–189. ACM, 2001. *Cited on pages:* 3 and 165.

[BKR00] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *J. Comput. Syst. Sci.*, 61(3):362–399, 2000. *Cited on page:* 3.

[BLT14] A Bogdanov, M Lauridsen, and E Tischhauser. Aes-based authenticated encryption modes in parallel high-performance software. DIAC presentation (2014), 2014. *Cited on page:* 260.

[BMOS17] Guy Barwell, Daniel P. Martin, Elisabeth Oswald, and Martijn Stam. Authenticated encryption in the face of protocol and side channel leakage. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 693–723. Springer, 2017. *Cited on page:* 163.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 531–545. Springer, 2000. *Cited on pages:* 172, 176, 182, and 224.

[BPS15] Guy Barwell, Daniel Page, and Martijn Stam. Rogue decryption failures: Reconciling AE robustness notions. In Jens Groth, editor, *Cryptography and Coding - 15th IMA International Conference, IMACC 2015, Oxford, UK, December 15-17, 2015. Proceedings*, volume 9496 of *Lecture Notes in Computer Science*, pages 94–111. Springer, 2015. *Cited on page:* 163.

[BR00] Mihir Bellare and Phillip Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 317–330. Springer, 2000. *Cited on pages:* 3 and 134.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, 2006. *Cited on pages:* 17 and 21.

[BS16] Raphael Bost and Olivier Sanders. Trick or tweak: On the (in)security of otr's tweaks. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 333–353, 2016. *Cited on page:* 196.

[BT04] Alexandra Boldyreva and Nut Taesombut. Online encryption schemes: New security notions and constructions. In Tatsuaki Okamoto, editor, *Topics in Cryptology - CT-RSA 2004, The Cryptographers' Track at the RSA Conference 2004, San Francisco, CA, USA, February 23-27, 2004, Proceedings*, volume 2964 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2004. *Cited on page:* 133.

[CDH$^+$] D Chang, M Dworkin, S Hong, J Kelsey, and M Nandi. A Keyed Sponge Construction with Pseudorandomness in the Standard Model. NIST SHA-3 2012 Workshop. `http://csrc.nist.gov/groups/ST/hash/`

`sha-3/Round3/March2012/documents/papers/CHANG_paper.pdf`. *Cited on pages:* 96 and 106.

[CG16] Colin Chaigneau and Henri Gilbert. Is AEZ v4.1 sufficiently resilient against key-recovery attacks? *IACR Trans. Symmetric Cryptol.*, 2016(1):114–133, 2016. *Cited on pages:* 195, 201, and 205.

[CHVV03] Brice Canvel, Alain P. Hiltgen, Serge Vaudenay, and Martin Vuagnoux. Password interception in a SSL/TLS channel. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 583–599. Springer, 2003. *Cited on page:* 3.

[CMN$^+$14] Simon Cogliani, Diana-Stefania Maimut, David Naccache, Rodrigo Portella do Canto, Reza Reyhanitabar, Serge Vaudenay, and Damian Vizár. OMD: A compression function mode of operation for authenticated encryption. In Antoine Joux and Amr M. Youssef, editors, *Selected Areas in Cryptography - SAC 2014 - 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers*, volume 8781 of *Lecture Notes in Computer Science*, pages 112–128. Springer, 2014. *Cited on page:* 33.

[Cor] Intel Corpor. Intel intrinsics guide. `https://software.intel.com/sites/landingpage/IntrinsicsGuide/`. *Cited on page:* 50.

[Cor18] Intel Corpor. Intel® architecture instruction set extensions and future features programming reference. `https://software.intel.com/sites/default/files/managed/c5/15/architecture-instruction-set-extensions-programming-reference.pdf`, 2018. *Cited on page:* 51.

[CS08] Debrup Chakraborty and Palash Sarkar. A general construction of tweakable block ciphers and different modes of operations. *IEEE Trans. Information Theory*, 54(5):1991–2006, 2008. *Cited on pages:* 37 and 49.

[CS14] Shan Chen and John P. Steinberger. Tight security bounds for key-alternating ciphers. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 327–350. Springer, 2014. *Cited on pages:* 17 and 18.

[CW77] Larry Carter and Mark N. Wegman. Universal classes of hash functions (extended abstract). In John E. Hopcroft, Emily P. Friedman, and Michael A. Harrison, editors, *Proceedings of the 9th Annual ACM Symposium* *on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, pages 106–112. ACM, 1977. *Cited on pages:* 3 and 37.

[Dam89] Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1989. *Cited on pages:* 33 and 36.

[Dam07] Ivan Damgård. A "proof-reading" of some issues in cryptography. In *International Colloquium on Automata, Languages, and Programming*, pages 2–11. Springer, 2007. *Cited on page:* 219.

[DEMS] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Remark on variable tag lengths and OMD. `crypto-competitions` mailing list. April 25, 2014. *Cited on pages:* 164 and 167.

[DEMS14] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1, March 2014. *Cited on page:* 96.

[DEMS16] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schl affer. Ascon v1.2. `https://competitions.cr.yp.to/round3/asconv12.pdf`, 2016. *Cited on pages:* 201, 225, 270, and 272.

[Des77] Des. Data encryption standard. In *In FIPS PUB 46, Federal Information Processing Standards Publication*, pages 46–2, 1977. *Cited on page:* 1.

[DG17] William Diehl and Kris Gaj. RTL implementations and FPGA benchmarking of selected CAESAR round two authenticated ciphers. *Microprocessors and Microsystems - Embedded Hardware Design*, 52:202–218, 2017. *Cited on page:* 42.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976. *Cited on page:* 2.

[Die16] William Diehl. Fpga implementation of omdv2.0 (rtl vhdl), jul 2016. *Cited on page:* 42.

[DMA17] Joan Daemen, Bart Mennink, and Gilles Van Assche. Full-state keyed duplex with built-in multi-user support. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 606–637. Springer, 2017. *Cited on page:* 97.

[dMGSP08] Giacomo de Meulenaer, François Gosset, François-Xavier Standaert, and Olivier Pereira. On the energy cost of communication and cryptography in wireless sensor networks. In *IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob 2008, Avignon, France, 12-14 October 2008, Proceedings*, pages 580–585. IEEE Computer Society, 2008. *Cited on page:* 165.

[DMV17] Joan Daemen, Bart Mennink, and Gilles Van Assche. Full-state keyed duplex with built-in multi-user support. *IACR Cryptology ePrint Archive*, 2017:498, 2017. *Cited on page:* 196.

[DN14a] Nilanjan Datta and Mridul Nandi. ELmD v1.0, March 2014. *Cited on page:* 260.

[DN14b] Nilanjan Datta and Mridul Nandi. Elme: A misuse resistant parallel authenticated encryption. In Willy Susilo and Yi Mu, editors, *Information Security and Privacy - 19th Australasian Conference, ACISP 2014, Wollongong, NSW, Australia, July 7-9, 2014. Proceedings*, volume 8544 of *Lecture Notes in Computer Science*, pages 306–321. Springer, 2014. *Cited on page:* 260.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Information Security and Cryptography. Springer, 2002. *Cited on pages:* 2, 266, and 272.

[DR11] Thai Duong and Juliano Rizzo. Here come the $\oplus$ ninjas. Unpublished manuscript, 2011. *Cited on page:* 138.

[Dro15] Johan Droz. Fast implementation of OMD mode. Student project at Ecole Polytechnique Fédéral Lausanne, 2015. *Cited on pages:* 33, 52, and 54.

[Dwo01] Morris Dworkin. Recommendation for block cipher modes of operation. methods and techniques. Technical report, NATIONAL INST OF STANDARDS AND TECHNOLOGY GAITHERSBURG MD COMPUTER SECURITY DIV, 2001. *Cited on page:* 3.

[Dwo04] M. Dworkin. Recommendation for block cipher modes of operation: the CCM mode for authentication and confidentiality. NIST Special Publication 800-38C, May 2004. *Cited on page:* 166.

[Dwo07] M. Dworkin. Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC. NIST Special Publication 800-38D, November 2007. *Cited on page:* 166.

[Dwo16] Morris J Dworkin. Recommendation for block cipher modes of operation: The cmac mode for authentication. Technical report, 2016. *Cited on page:* 3.

[EBFK13] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. An empirical study of cryptographic misuse in Android applications. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 73–84. ACM, 2013. *Cited on page:* 165.

[EM91] Shimon Even and Yishay Mansour. A construction of a cioher from a single pseudorandom permutation. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Advances in Cryptology - ASIACRYPT '91, International Conference on the Theory and Applications of Cryptology, Fujiyoshida, Japan, November 11-14, 1991, Proceedings*, volume 739 of *Lecture Notes in Computer Science*, pages 210–224. Springer, 1991. *Cited on pages:* 96 and 105.

[EM97] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. *J. Cryptology*, 10(3):151–162, 1997. *Cited on pages:* 96 and 105.

[EV16] Guillaume Endignoux and Damian Vizár. Linking online misuse-resistant authenticated encryption and blockwise attack models. *IACR Trans. Symmetric Cryptol.*, 2016(2):125–144, 2016. *Cited on page:* 132.

[Fer02] N Ferguson. Collision attacks on ocb. *NIST CSRC website*, 2002. *Cited on pages:* 195, 201, and 206.

[Fer05] Niels Ferguson. Authentication weaknesses in GCM. *Comments submitted to NIST Modes of Operation Process*, 2005. *Cited on page:* 4.

[FFL12] Ewan Fleischmann, Christian Forler, and Stefan Lucks. Mcoe: A family of almost foolproof on-line authenticated encryption schemes. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 196–215. Springer, 2012. *Cited on pages:* 6, 9, 132, 134, 135, 136, 163, 164, 165, 198, 199, 221, 259, 260, and 262.

[FJMV03] Pierre-Alain Fouque, Antoine Joux, Gwenaëlle Martinet, and Frédéric Valette. Authenticated on-line encryption. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography, 10th Annual International Workshop, SAC 2003, Ottawa, Canada, August 14-15, 2003, Revised Papers*, volume 3006 of *Lecture Notes in Computer Science*, pages 145–159. Springer, 2003. *Cited on pages:* 9 and 132.

[FLLW17] Christian Forler, Eik List, Stefan Lucks, and Jakob Wenzel. Reforgeability of authenticated encryption schemes. In *Information Security and Privacy - 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July*

3-5, 2017, Proceedings, Part II, volume 10343, pages 19–37. Springer, 2017. *Cited on page:* 196.

[FLS15] Thomas Fuhr, Gaëtan Leurent, and Valentin Suder. Collision attacks against CAESAR candidates - forgery and key-recovery against AEZ and marble. In *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 510–532. Springer, 2015. *Cited on page:* 195.

[Fou18] OpenSSL Software Foundation. Openssl cryptography and ssl/tls toolkit. `https://www.openssl.org/source/`, 2018. *Cited on page:* 50.

[GCG12] Jim Guilford, David Cote, and Vinodh Gopal. Fast SHA512 Implementations on Intel® Architecture Processors, nov 2012. *Cited on pages:* 34 and 51.

[Geo15] Martin Georgiev. Implementation of OMD mode on MIPS architecture. Student project at Ecole Polytechnique Fédéral Lausanne, 2015. *Cited on pages:* 33, 54, and 55.

[GGY$^+$13] Sean Gulley, Vinodh Gopal, Kirk Yap, Wajdi Feghali, Jim Guilford, and Gil Wolrich. Intel® sha extensions: New instructions supporting the secure hash algorithm on intel® architecture processors. `https://software.intel.com/sites/default/files/article/402097/intel-sha-extensions-white-paper.pdf`, jul 2013. *Cited on pages:* 34, 51, and 58.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 365–377. ACM, 1982. *Cited on page:* 2.

[GMS$^+$14] Danilo Gligoroski, Hristina Mihajloska, Simona Samardjiska, Håkon Jacobsen, Mohamed El-Hadedy, Rune Erlend Jensen, and Daniel Otte. CAESAR submission: Keyak v1, March 2014. *Cited on page:* 96.

[GPT15] Peter Gazi, Krzysztof Pietrzak, and Stefano Tessaro. The exact PRF security of truncation: Tight bounds for keyed sponges and truncated CBC. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture*

Notes in Computer Science, pages 368–387. Springer, 2015. *Cited on page:* 96.

[Guo14a] J Guo. Caesar candidate marble. DIAC presentation (2014), 2014. *Cited on page:* 260.

[Guo14b] Jian Guo. Marble Specification Version 1.0, March 2014. *Cited on page:* 260.

[GYG12] Jim Guilford, Kirk Yap, and Vinodh Gopal. Fast sha-256 implementations on intel® architecture processors. `http://www.intel.com/content/www/us/en/intelligent-systems/intel-technology/sha-256-implementations-paper.html`, may 2012. *Cited on pages:* 34 and 51.

[HKR15] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. Robust authenticated-encryption: AEZ and the problem that it solves. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 15–44. Springer, 2015. *Cited on pages:* 5, 147, 156, 157, 163, 164, 182, 198, 199, 224, and 263.

[HKR17] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. AEZ v5: Authenticated encryption by enciphering. `https://competitions.cr.yp.to/round3/aezv5.pdf`, 2017. *Cited on pages:* 201, 205, and 266.

[Hot10] George Hotz. Console hacking 2010-ps3 epic fail. In *27th Chaos Communications Congress*, 2010. *Cited on page:* 165.

[HP08] Helena Handschuh and Bart Preneel. Key-recovery attacks on universal hash function based MAC algorithms. In *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*, pages 144–161. Springer, 2008. *Cited on page:* 195.

[HRRV15] Viet Tung Hoang, Reza Reyhanitabar, Phillip Rogaway, and Damian Vizár. Online authenticated-encryption and its nonce-reuse misuse-resistance. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 493–517. Springer, 2015. *Cited on pages:* 131, 136, 163, and 164.

[HW14] Tao Huang and Hongjun Wu. Attack on AES-OTR. `crypto-competitions` mailing list. March 21, 2014., 2014. *Cited on page:* 196.

[IMG$^+$14a] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, Sumio Morioka, and Eita Kobayashi. CLOC: Compact Low-Overhead CFB. `https://competitions.cr.yp.to/round1/clocv1.pdf`, 2014. *Cited on page:* 260.

[IMG$^+$14b] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, Sumio Morioka, and Eita Kobayashi. SILC: SImple Lightweight CFB. `https://competitions.cr.yp.to/round1/silcv1.pdf`, 2014. *Cited on page:* 260.

[IMG$^+$16] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, Sumio Morioka, and Eita Kobayashi. CLOC and SILC. `https://competitions.cr.yp.to/round3/clocsilcv3.pdf`, 2016. *Cited on pages:* 201 and 267.

[IOM12] Tetsu Iwata, Keisuke Ohashi, and Kazuhiko Minematsu. Breaking and repairing GCM security proofs. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 31–49. Springer, 2012. *Cited on pages:* 4, 195, and 204.

[Iwa06] Tetsu Iwata. New blockcipher modes of operation with beyond the birthday bound security. In Matthew J. B. Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*, pages 310–327. Springer, 2006. *Cited on pages:* 9 and 35.

[Iwa08] Tetsu Iwata. Authenticated encryption mode for beyond the birthday bound security. In Serge Vaudenay, editor, *Progress in Cryptology - AFRICACRYPT 2008, First International Conference on Cryptology in Africa, Casablanca, Morocco, June 11-14, 2008. Proceedings*, volume 5023 of *Lecture Notes in Computer Science*, pages 125–142. Springer, 2008. *Cited on page:* 35.

[Iwa15] Tetsu Iwata. CLOC and SILC will be tweaked. `crypto-competitions` mailing list. August 4, 2015., 2015. *Cited on pages:* 164, 165, and 166.

[IY09a] Tetsu Iwata and Kan Yasuda. BTM: A single-key, inverse-cipher-free mode for deterministic authenticated encryption. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers*, volume 5867 of *Lecture Notes in Computer Science*, pages 313–330. Springer, 2009. *Cited on pages:* 57 and 63.

[IY09b] Tetsu Iwata and Kan Yasuda. HBS: A single-key mode of operation for deterministic authenticated encryption. In Orr Dunkelman, editor, *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers*, volume 5665 of *Lecture Notes in Computer Science*, pages 394–415. Springer, 2009. *Cited on pages:* 57 and 63.

[JLM14] Philipp Jovanovic, Atul Luykx, and Bart Mennink. Beyond 2 c/2 security in sponge-based authenticated encryption modes. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 85–104. Springer, 2014. *Cited on page:* 96.

[JNP16] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Deoxys v1.41. `https://competitions.cr.yp.to/round3/deoxysv141.pdf`, 2016. *Cited on pages:* 201 and 267.

[JNPS14a] Jérémy Jean, Ivica Nikolić, Thomas Peyrin, and Yannick Seurin. Deoxys v1, March 2014. *Cited on page:* 260.

[JNPS14b] Jérémy Jean, Ivica Nikolić, Thomas Peyrin, and Yannick Seurin. Joltik v1, March 2014. *Cited on page:* 260.

[JNPS14c] Jérémy Jean, Ivica Nikolić, Thomas Peyrin, and Yannick Seurin. KIASU v1, March 2014. *Cited on page:* 260.

[Jou06] Antoine Joux. Authentication failures in nist version of GCM. "`https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/comments/800-38-series-drafts/gcm/joux_comments.pdf`", 2006. *Cited on pages:* 4, 195, 201, and 204.

[Jut01] Charanjit S. Jutla. Encryption modes with almost free message integrity. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*, pages 529–544. Springer, 2001. *Cited on page:* 4.

[Kah96] David Kahn. *The codebreakers.* Scribner, 1996. *Cited on page:* 1.

[KEM17] Daniel Kales, Maria Eichlseder, and Florian Mendel. Note on the robustness of caesar candidates. Cryptology ePrint Archive, Report 2017/1137, 2017. `https://eprint.iacr.org/2017/1137`. *Cited on pages:* 196 and 197.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014. *Cited on page:* 2.

[KLL$^+$14] Elif Bilge Kavun, Martin M. Lauridsen, Gregor Leander, Christian Rechberger, Peter Schwabe, and Tolga Yalçın. Prøst v1, March 2014. *Cited on pages:* 96 and 260.

[KM04] Neal Koblitz and Alfred Menezes. Another look at "provable security". Cryptology ePrint Archive, Report 2004/152, 2004. `https://eprint.iacr.org/2004/152`. *Cited on page:* 219.

[KM06] Neal Koblitz and Alfred Menezes. Another look at "provable security". ii. Cryptology ePrint Archive, Report 2006/229, 2006. `https://eprint.iacr.org/2006/229`. *Cited on page:* 219.

[KM11] Neal Koblitz and Alfred Menezes. Another Look at Provable Security. "`http://anotherlook.ca/`", 2011. *Cited on page:* 219.

[KR11] Ted Krovetz and Phillip Rogaway. The software performance of authenticated-encryption modes. In Antoine Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*, pages 306–327. Springer, 2011. *Cited on pages:* 33, 37, 49, 52, 82, 90, 164, 182, 183, 184, 187, 188, 189, 191, and 254.

[KR14] Ted Krovetz and Phillip Rogaway. The ocb authenticated-encryption algorithm. Technical report, RFC Editor, 2014. *Cited on page:* 164.

[KR16] Ted Krovetz and Phillip Rogaway. OCB (v1.1). `https://competitions.cr.yp.to/round3/ocbv11.pdf`, 2016. *Cited on pages:* 201 and 266.

[KY00] Jonathan Katz and Moti Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In Bruce Schneier, editor, *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*, volume 1978 of *Lecture Notes in Computer Science*, pages 284–299. Springer, 2000. *Cited on pages:* 3 and 134.

[lab08] VAMPIRE lab. System for unified performance evaluation related to cryptographic operations and primitives. `https://bench.cr.yp.to/supercop.html`, 2008. *Cited on page:* 55.

[lab18] VAMPIRE lab. Measurements of authenticated ciphers, indexed by machine. `https://bench.cr.yp.to/results-aead.html`, 2018. *Cited on page:* 55.

[Lan14] Adam Langley. Apple's ssl/tls bug. *Imperial Violet*, 2014. *Cited on page:* 165.

[LPRM07] David Lefranc, Philippe Painchault, Valérie Rouat, and Emmanuel Mayer. A generic method to design modes of operation beyond the birthday bound. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers*, volume 4876 of *Lecture Notes in Computer Science*, pages 328–343. Springer, 2007. *Cited on page:* 35.

[LRW02] Moses Liskov, Ronald L. Rivest, and David A. Wagner. Tweakable block ciphers. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2002. *Cited on page:* 20.

[LST12] Will Landecker, Thomas Shrimpton, and R. Seth Terashima. Tweakable blockciphers with beyond birthday-bound security. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 14–30. Springer, 2012. *Cited on page:* 35.

[Lu17] Jiqiang Lu. Almost universal forgery attacks on the COPA and marble authenticated encryption algorithms. In Ramesh Karri, Ozgur Sinanoglu, Ahmad-Reza Sadeghi, and Xun Yi, editors, *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017*, pages 789–799. ACM, 2017. *Cited on page:* 196.

[Luc14] S Lucks. Personal communication, 2014. *Cited on page:* 136.

[LZLG14] Yong Li, Yuanyuan Zhang, Juanru Li, and Dawu Gu. icryptotracer: Dynamic analysis on misuse of cryptography functions in iOS applications. In *Network and System Security - 8th International Conference, NSS 2014, Xi'an, China, October 15-17, 2014, Proceedings*, volume 8792 of *Lecture Notes in Computer Science*, pages 349–362. Springer, 2014. *Cited on page:* 165.

[Man] James H. Manger. [CFRG] Attacker changing tag length in OCB. IRTF `CFRG` mailing list. May 29, 2013. *Cited on pages:* 164, 165, 166, and 170.

[MDV16] Aleksandra Mileva, Vesna Dimitrova, and Vesselin Velichkov. Analysis of the authenticated cipher MORUS (v1). In *Cryptography and Information Security in the Balkans - Second International Conference, BalkanCryptSec 2015, Koper, Slovenia, September 3-4, 2015, Revised Selected Papers*, volume 9540 of *Lecture Notes in Computer Science*, pages 45–59. Springer, 2016. *Cited on page:* 196.

[Mer89] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer, 1989. *Cited on pages:* 33 and 36.

[Mer09] Merriam-Webster Online. Merriam-Webster Online Dictionary, 2009. *Cited on page:* 1.

[MGH+14] Paweł Morawiecki, Kris Gaj, Ekawat Homsirikamol, Krystian Matusiewicz, Josef Pieprzyk, Marcin Rogawski, Marian Srebrny, and Marcin Wójcik. ICEPOLE v1, March 2014. *Cited on pages:* 96 and 260.

[Mia14] W Miaw. Netflix / msl. `https://github.com/Netflix/msl/wiki`, 2014. *Cited on page:* 131.

[Min14a] Kazuhiko Minematsu. AES-OTR v1. `https://competitions.cr.yp.to/round1/aesotrv1.pdf`, 2014. *Cited on page:* 260.

[Min14b] Kazuhiko Minematsu. Parallelizable rate-1 authenticated encryption from pseudorandom functions. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 275–292. Springer, 2014. *Cited on page:* 260.

[Min15] Kazuhiko Minematsu. AES-OTR v2. `crypto-competitions` mailing list. August 31, 2015., 2015. *Cited on pages:* 164 and 166.

[Min16] Kazuhiko Minematsu. AES-OTR v3.1. `https://competitions.cr.yp.to/round3/aesotrv31.pdf`, 2016. *Cited on pages:* 201 and 267.

[MMH+14] Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel, and Ingrid Verbauwhede. Chaskey: An efficient MAC algorithm for 32-bit microcontrollers. In Antoine Joux and Amr M. Youssef, editors, *Selected Areas in Cryptography - SAC 2014 - 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers*, volume 8781 of *Lecture Notes in Computer Science*, pages 306–323. Springer, 2014. *Cited on page:* 96.

[MRH04] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2004. *Cited on page:* 96.

[MRV15] Bart Mennink, Reza Reyhanitabar, and Damian Vizár. Security of full-state keyed sponge and duplex: Applications to authenticated encryption. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 465–489. Springer, 2015. *Cited on page:* 95.

[MV04] David A. McGrew and John Viega. The security and performance of the galois/counter mode (GCM) of operation. In *INDOCRYPT 2004*, pages 343–355, 2004. *Cited on pages:* 4, 166, 201, and 265.

[MvOV96] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996. *Cited on page:* 3.

[Nan] Mridul Nandi. RE:CLOC and SILC will be tweaked. `crypto-competitions` mailing list. August 5, 2015. *Cited on pages:* 167 and 168.

[Nik16] Ivica Nikolić. Tiaoxin - 346. `https://competitions.cr.yp.to/round3/tiaoxinv21.pdf`, 2016. *Cited on pages:* 201, 210, and 268.

[NRS13] Chanathip Namprempre, Phillip Rogaway, and Tom Shrimpton. AE5 security notions: Definitions implicit in the CAESAR call. *IACR Cryptology ePrint Archive*, 2013:242, 2013. *Cited on page:* 4.

[NRS14] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. Reconsidering generic composition. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 257–274. Springer, 2014. *Cited on page:* 57.

[oST12] National Institute of Standards and Technology. Secure hash standard (SHS). NIST FIPS PUB 180-4, mar 2012. *Cited on pages:* 33, 35, and 40.

[oST15] National Institute of Standards and Technology. SHA-3 standard: Permutation-based hash and extendable-output functions. Federal Inf. Process. Stds. (NIST FIPS) - 202, aug 2015. *Cited on page:* 95.

[Par16] Kevin Parrish. Intel says that 'apollo lake' processors will ship in the second half of 2016. `https://www.digitaltrends.com/computing/apollo-lake-intel/`, 2016. *Cited on pages:* 51 and 58.

[Pat08a] Jacques Patarin. The "coefficients h" technique. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers*, volume 5381 of *Lecture Notes in Computer Science*, pages 328–345. Springer, 2008. *Cited on page:* 18.

[Pat08b] Jacques Patarin. A proof of security in o(2n) for the xor of two random permutations. In Reihaneh Safavi-Naini, editor, *Information Theoretic Security, Third International Conference, ICITS 2008, Calgary, Canada, August 10-13, 2008, Proceedings*, volume 5155 of *Lecture Notes in Computer Science*, pages 232–248. Springer, 2008. *Cited on page:* 9.

[Pat10] Jacques Patarin. Introduction to mirror theory: Analysis of systems of linear equalities and linear non equalities for cryptography. *IACR Cryptology ePrint Archive*, 2010:287, 2010. *Cited on page:* 9.

[PC15] Gordon Procter and Carlos Cid. On weak keys and forgery attacks against polynomial-based MAC schemes. *J. Cryptology*, 28(4):769–795, 2015. *Cited on page:* 195.

[Per14] Ray Perlner. Extendable-Output Functions (XOFs). NIST SHA-3 2014 Workshop, 2014. `http://csrc.nist.gov/groups/ST/hash/sha-3/Aug2014/documents/perlner_XOFs.pdf`. *Cited on page:* 96.

[Pub01] NIST FIPS Pub. 197: Advanced encryption standard (aes). *Federal information processing standards publication*, 197(441):0311, 2001. *Cited on page:* 2.

[RBBK01] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In Michael K. Reiter and Pierangela Samarati, editors, *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001.*, pages 196–205. ACM, 2001. *Cited on pages:* 4, 33, 82, 134, 164, and 183.

[Rec14] Francisco Recacha. ++AE v1.0, March 2014. *Cited on page:* 260.

[Rey] Reza Reyhanitabar. OMD version 2: a tweak for the 2nd round. `crypto-competitions` mailing list. August 27, 2015. *Cited on pages:* 164 and 166.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 98–107. ACM, 2002. *Cited on pages:* 4, 5, 22, 23, 24, 33, 75, 164, 167, 176, 203, and 221.

[Rog04a] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004. *Cited on pages:* 33, 37, 48, 49, 63, 82, 90, 164, 179, 183, 187, and 254.

[Rog04b] Phillip Rogaway. Nonce-based symmetric encryption. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2004. *Cited on page:* 164.

[Rog13] Phillip Rogaway. Re: [CFRG] Attacker changing tag length in OCB. IRTF `CFRG` mailing list. Jun 3, 2013., 2013. *Cited on pages:* 166, 167, 168, and 170.

[RS06a] Phillip Rogaway and Thomas Shrimpton. Deterministic authenticated-encryption: A provable-security treatment of the key-wrap problem. *IACR Cryptology ePrint Archive*, 2006:221, 2006. *Cited on page:* 69.

[RS06b] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390. Springer, 2006. *Cited on pages:* 4, 5, 24, 25, 26, 27, 28, 29, 57, 60, 63, 131, 132, 154, 156, 163, 164, 165, 176, 177, 182, 198, 199, 221, 224, and 259.

[RS14] Ronald L. Rivest and Jacob C. N. Schuldt. Spritz – a Spongy RC4-like Stream Cipher and Hash Function, 2014. `https://people.csail.mit.edu/rivest/pubs/RS14.pdf`. *Cited on page:* 96.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978. *Cited on page:* 2.

[RVV14] Reza Reyhanitabar, Serge Vaudenay, and Damian Vizár. Misuse-resistant variants of the OMD authenticated encryption mode. In Sherman S. M. Chow, Joseph K. Liu, Lucas Chi Kwong Hui, and Siu-Ming Yiu, editors, *Provable Security - 8th International Conference, ProvSec 2014, Hong Kong, China, October 9-10, 2014. Proceedings*, volume 8782 of *Lecture Notes in Computer Science*, pages 55–70. Springer, 2014. *Cited on page:* 57.

[RVV15] Reza Reyhanitabar, Serge Vaudenay, and Damian Vizár. Boosting OMD for almost free authentication of associated data. In Gregor Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 411–427. Springer, 2015. *Cited on pages:* 75 and 97.

[RVV16] Reza Reyhanitabar, Serge Vaudenay, and Damian Vizár. Authenticated encryption with variable stretch. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 396–425, 2016. *Cited on page:* 163.

[RW03] Phillip Rogaway and David Wagner. A Critique of CCM. *IACR Cryptology ePrint Archive*, 2003:70, 2003. *Cited on pages:* 4, 166, and 170.

[RZ11] Phillip Rogaway and Haibin Zhang. Online ciphers from tweakable blockciphers. In Aggelos Kiayias, editor, *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, volume 6558 of *Lecture Notes in Computer Science*, pages 237–249. Springer, 2011. *Cited on page:* 142.

[Saa12] Markku-Juhani Olavi Saarinen. Cycling attacks on GCM, GHASH and other polynomial MACs and hashes. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 216–225. Springer, 2012. *Cited on pages:* 4 and 195.

[Saa14] Markku-Juhani O. Saarinen. The CBEAMr1 Authenticated Encryption Algorithm, March 2014. *Cited on page:* 260.

[SB14] Markku-Juhani O. Saarinen and Billy B. Brumley. The STRIBOBr1 Authenticated Encryption Algorithm, March 2014. *Cited on pages:* 96 and 260.

[Sha45] Claude E. Shannon. A mathematical theory of cryptography. *Bell Systems Technical Journal*, 1945. *Cited on pages:* 1 and 2.

[Sha49] Claude E Shannon. Communication theory of secrecy systems. *Bell Labs Technical Journal*, 28(4):656–715, 1949. *Cited on page:* 1.

[Sim84] Gustavus J. Simmons. Authentication theory/coding theory. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984,* *Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 411–431. Springer, 1984. *Cited on page:* 1.

[ST17] Pratik Soni and Stefano Tessaro. Public-seed pseudorandom permutations. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 412–441, 2017. *Cited on pages:* 101 and 219.

[STA^{+}14] Yu Sasaki, Yosuke Todo, Kazumaro Aoki, Yusuke Naito, Takeshi Sugawara, Yumiko Murakami, and Mitsuru Matsui nad Shoichi Hirose. Minalpher v1, March 2014. *Cited on page:* 260.

[Sti95] Douglas R. Stinson. *Cryptography - theory and practice.* Discrete mathematics and its applications series. CRC Press, 1995. *Cited on page:* 1.

[Str] R. Struik. AEAD Ciphers for Highly Constrained Networks. DIAC 2013 presentation, August 13, 2013. *Cited on page:* 165.

[SWZ13] Zhelei Sun, Peng Wang, and Liting Zhang. Collision attacks on variant of OCB mode and its series. In *Information Security and Cryptology - 8th International Conference, Inscrypt 2012, Beijing, China, November 28-30, 2012, Revised Selected Papers*, volume 7763 of *Lecture Notes in Computer Science*, pages 216–224. Springer, 2013. *Cited on page:* 195.

[SY15] Yu Sasaki and Kan Yasuda. How to incorporate associated data in sponge-based authenticated encryption. In Kaisa Nyberg, editor, *Topics in Cryptology - CT-RSA 2015, The Cryptographer's Track at the RSA Conference 2015, San Francisco, CA, USA, April 20-24, 2015. Proceedings*, volume 9048 of *Lecture Notes in Computer Science*, pages 353–370. Springer, 2015. *Cited on page:* 97.

[TSS09] Patrick P Tsang, Rouslan V Solomakhin, and Sean W Smith. Authenticated streamwise on-line encryption. Dartmouth Computer Science Technical Report TR2009-640, 2009. *Cited on page:* 133.

[Vau02] Serge Vaudenay. Security flaws induced by CBC padding - applications to ssl, ipsec, WTLS ... In Lars R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*, pages 534–546. Springer, 2002. *Cited on page:* 3.

[Viz16] Damian Vizár. The state of the authenticated encryption. *Tatra Mountains Mathematical Publications*, 67(1):167–190, 2016. *Cited on pages:* 5 and 131.

[VV17] Serge Vaudenay and Damian Vizár. Under pressure: Security of caesar candidates beyond their guarantees. Cryptology ePrint Archive, Report 2017/1147, 2017. https://eprint.iacr.org/2017/1147. *Cited on page:* 195.

[Wan14] Lei Wang. SHELL v1, March 2014. *Cited on page:* 260.

[WC81] Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981. *Cited on pages:* 3 and 37.

[WH14a] Hongjun Wu and Tao Huang. The authenticated cipher MORUS (v1). https://competitions.cr.yp.to/round1/morusv1.pdf, 2014. *Cited on page:* 260.

[WH14b] Hongjun Wu and Tao Huang. The JAMBU lightweight authentication encryption mode (v1). https://competitions.cr.yp.to/round1/jambuv1.pdf, 2014. *Cited on page:* 260.

[WH16a] Hongjun Wu and Tao Huang. The authenticated cipher MORUS (v2). https://competitions.cr.yp.to/round3/morusv2.pdf, 2016. *Cited on pages:* 201, 225, 269, and 271.

[WH16b] Hongjun Wu and Tao Huang. The JAMBU lightweight authentication encryption mode (v2.1). https://competitions.cr.yp.to/round3/jambuv21.pdf, 2016. *Cited on pages:* 201 and 273.

[WHF03a] D. Whiting, R. Housley, and N. Ferguson. Counter with CBC-MAC (CCM). IETF RFC 3610 (Informational), September 2003. *Cited on page:* 166.

[WHF03b] Doug Whiting, Russell Housley, and Niels Ferguson. Counter with CBC-MAC (CCM). *RFC*, 3610:1–26, 2003. *Cited on pages:* 4, 201, and 265.

[WP16] Hongjun Wu and Bart Preneel. AEGIS: A fast authenticated encryption algorithm (v1.1). https://competitions.cr.yp.to/round3/aegisv11.pdf, 2016. *Cited on pages:* 196, 201, 211, and 268.

[Wu05] Hongjun Wu. The misuse of RC4 in microsoft word and excel. *IACR Cryptology ePrint Archive*, 2005:7, 2005. *Cited on page:* 165.

[Wu16] Hongjun Wu. ACORN: A lightweight authenticated cipher (v3). https://competitions.cr.yp.to/round2/acornv2.pdf, 2016. *Cited on pages:* 201 and 268.

[Yas07] Kan Yasuda. Boosting Merkle-Damgård hashing for message authentication. In Kaoru Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application* *of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*, pages 216–231. Springer, 2007. *Cited on pages:* 75 and 76.

[ZWHS14] Liting Zhang, Wenling Wu, and Peng Wang Han Sui. iFeed[AES] v1. `https://competitions.cr.yp.to/round1/ifeedaesv1.pdf`, 2014. *Cited on page:* 260.

Appendix A

Performance and Security of Compression Function-based AE

A.1 Performance of OMD in Software

We provide a visualization of the complete sets of performance measurements for the reference implementation (Figures A.1 and A.4), SSE4-based implementation (Figures A.2 and A.5) and AVX1-based implementation (Figures A.3 and A.6) of OMD-sha256 and OMD-sha512 carried out on a 64-bit 2.4GHz dual-core Intel Core i5-2415M processor running Ubuntu 12.10.

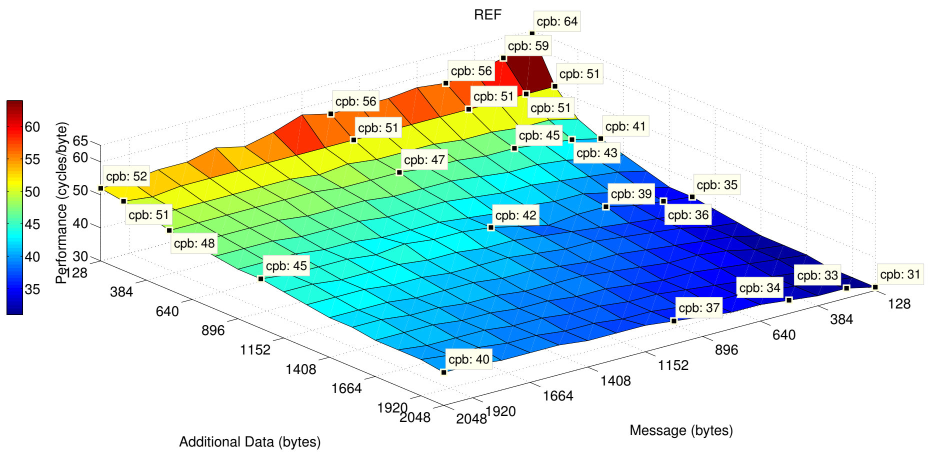


Figure A.1 – **3D surface plot of the performance of the OMD-sha256 reference implementation** with increasing message and AD length. Graph from Ankele and Ankele [AA14].

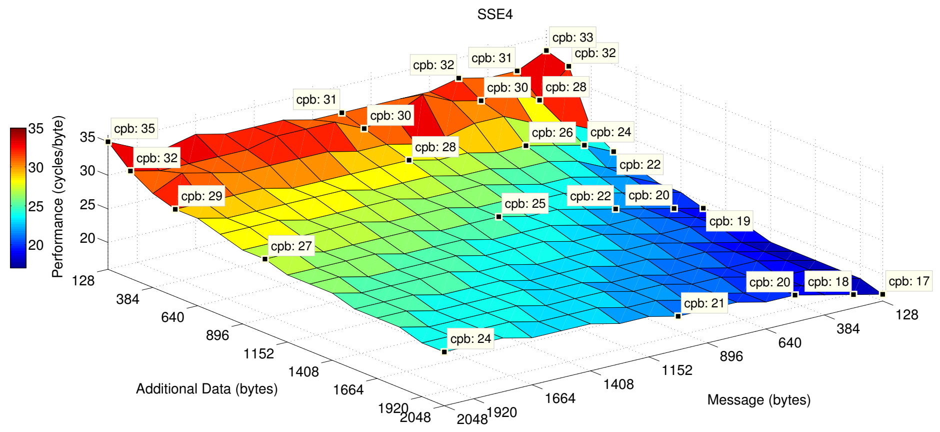


Figure A.2 – **3D surface plot of the performance of the OMD-sha256 SSE4 implementation** with increasing message and AD length. Graph from Ankele and Ankele [AA14].

A.2 The Rational behind the Masking Sequence $\Delta_{N,i,j}$

In this section, we explain the design of the masking function $\Delta_K(N, i, j)$ in p-OMD and show that it fulfils the required security properties.

p-OMD uses the XE construction [Rog04a] to instantiate a (tweakable) PRF $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ using a regular PRF $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ by defining $\widetilde{F}_K^{\mathsf{T}}(X, Y) = F_K((X \oplus \Delta_K(\mathsf{T})), Y)$ for every $\mathsf{T} \in \mathcal{T}, K \in \mathcal{K}$, $X \in \{0,1\}^n, Y \in \{0,1\}^m$ where $\mathcal{T} = \{0,1\}^\nu \times \mathbb{N}^+ \times \mathbb{N}^+$ and $\Delta_K(\mathsf{T})$ is the masking function of p-OMD.

The purpose of the masking function is to compute *masking offsets* $\Delta_{N,i,j}$ that are tweak-dependent and key-dependent in such a way that

1. $\Delta_K(\cdot)$ is a 2^{-n}-uniform 2^{-n}-AXU hash,
2. the masking offsets should be computable *efficiently* in the order they appear in the scheme.

We need to efficiently compute $\Delta_{N,i+1,j}$ if we have previously computed $\Delta_{N,i,j}$ for $0 \leq i$ (i.e. "increment" the i component), and also we need to efficiently compute $\Delta_{N,i,j'}$ if we have previously computed $\Delta_{N,i,j}$ for any $j, j' \in \{0, \ldots, 15\}$ (i.e. "switch" the j component). To achieve this, we adapt the approach based on standard Gray Code sequence from [KR11], described in the proof of Lemma 3.4. We repeat the definition for the convenience of the reader.

Gray Code sequence. For a fixed positive $r > 0$ the Gray Code sequence is a special ordering $a : \{0, 1, \cdots, 2^r - 1\} \to \{0,1\}^r$ of the set or r bit strings. It can be defined recursively as $\gamma_0 = 0^r$ and $\gamma_i = \gamma_{i-1} \oplus 2^{\mathtt{ntz}(i)}$ if $i \geq 1$, where $\mathtt{ntz}(i)$ denotes the number

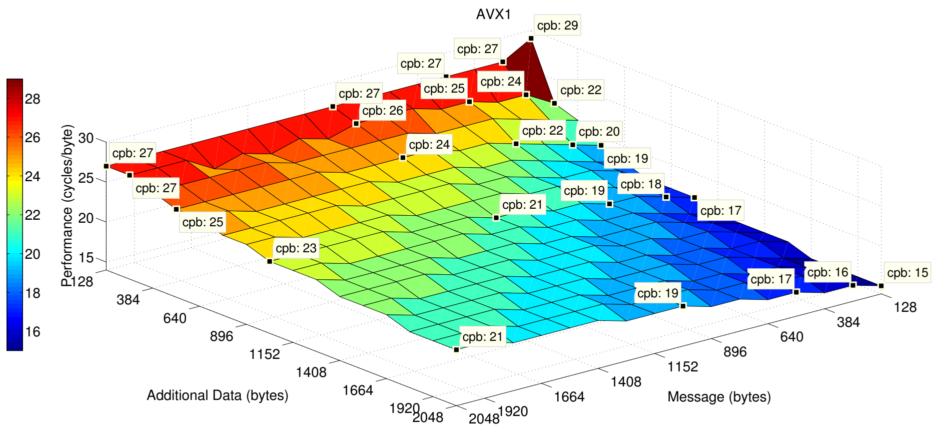


Figure A.3 – **3D surface plot of the performance of the OMD-sha256 AVX1 implementation** with increasing message and AD length. Graph from Ankele and Ankele [AA14].

of trailing zeros in the binary representation of i. The basic facts are that a is a bijection and $0 \leq \gamma_i \leq 2i$ (if represented as integer) for all i. We stress that we therefore have that (1) $\gamma_i \leq 2^{r+1}$ for all i, and (2) $\gamma_i \neq \gamma_j$ for all $i \neq j$.

Construction of the masking function and security properties. First recall that we only need to use 16 different values of the j component in p-OMD, i.e., all of its values are representable with 4 bits. Keeping this in mind, we first define the sequence of Galois field elements $\Gamma(i,j) \in GF(2^n)$ as $\Gamma(i,j) = 2^4 \cdot \gamma_i \oplus j$ for $0 \leq j \leq 15$ (we represent j by as an n bit string) and $0 \leq i < 2^{n-6}$ where γ_i is the i^{th} word of the canonical Gray code and the multiplications are in the Galois field. Referring to the properties of the Gray Code sequence, we can verify that for all i, we have $\mathsf{left}_5\left(\gamma_i\right) = 0^5$ and $\mathsf{right}_4\left(2^4 \cdot \gamma_i\right) = 0^4$. This implies that for every allowed pair i, j the value $\Gamma(i,j)$ will be a unique element of $GF(2^n)$.

Finally, we define the masking function as

$$\Delta_K(N,i,j) = F_K\left(N||10^n - |N| - 1, 0^m\right) \oplus \Gamma(i,j) \cdot F_K\left(0^n, 0^m\right)$$

where F is the PRF used in p-OMD.

We can now easily verify that the required security properties are met under the assumption that F is a good PRF. $\Delta_K(N,i,j)$ being a bitwise xor of two independent random n bit strings, we trivially have $\Pr[\Delta_K(N,i,j) = H] \leq 2^{-n}$ for any (N,i,j).

To see if $\Pr[\Delta_K(N,i,j) \oplus \Delta_K(N',i',j') = H] \leq 2^{-n}$ for $(N,i,j) \neq (N',i',j')$, we consider two cases, either $N = N'$ or not. In the latter case, $N \neq N'$ implies that $\Delta_K(N,i,j) \oplus \Delta_K(N',i',j') = H$ is equivalent to the event that a bitwise xor of two independent random n-bit strings is equal to some specific value and we conclude that the required property is verified in this case. In the former case, $F_K(N||10^n - |N| - 1, 0^m) =$

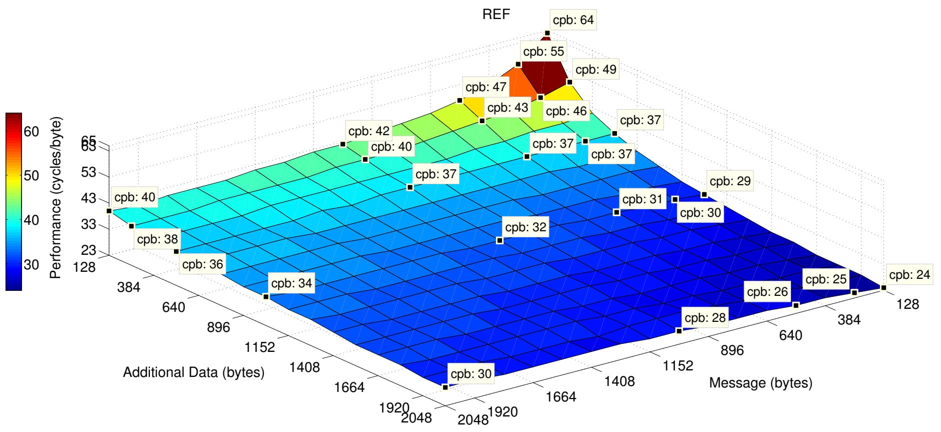


Figure A.4 – **3D surface plot of the performance of the OMD-sha512 reference implementation** with increasing message and AD length. Graph from Ankele and Ankele [AA14].

$F_K(N'||10^n-|N'|-1, 0^m)$ so $\Delta_K(N,i,j) \oplus \Delta_K(N',i',j') = H$ occurs iff $(\Gamma(i,j) \oplus \Gamma(i',j')) \cdot F_K(0^n, 0^m) = H$. Note that we must have $(i,j) \neq (i',j')$ which together with properties listed above imply that the multiplier $(\Gamma(i,j) \oplus \Gamma(i',j'))$ is non-zero. Thus, we conclude that the second condition is met in this case as well.

Compact representation. Let $L_* = F_K(0^n, 0^m)$. Then we have

$$\begin{aligned}\Delta_K(N,i,j) &= F_K\left(N||10^n - |N| - 1, 0^m\right) \oplus \Gamma(i,j) \cdot L_* \\ &= F_K\left(N||10^n - |N| - 1, 0^m\right) \oplus \gamma_i \cdot 2^4 \cdot L_* \oplus j \cdot L_*.\end{aligned}$$

We further define $L_*[j] = j \cdot L_*$ for $0 \leq j \leq 15$ and $L[\ell] = 2^{4+\ell} \cdot L_*$ for $0 \leq \ell < n-6$. Note that $L_*[1] = L_*$, $L_*[0] = 0^n$ and $L[0] = 2^4 \cdot L_*$. Thus we can write $\Delta_K(N,i,j) = F_K(N||10^n - |N| - 1, 0^m) \oplus \gamma_i \cdot L[0] \oplus L_*[j]$. We can derive two rules. First, keeping in mind the way we have defined the Gray Code sequence we can see that

$$\begin{aligned}\Delta_K(N,i,j) &= F_K\left(N||10^n - |N| - 1, 0^m\right) \oplus \gamma_i \cdot L[0] \oplus L_*[j] \\ &= F_K\left(N||10^n - |N| - 1, 0^m\right) \oplus \left(\gamma_{i-1} \oplus 2^{\texttt{ntz}(i)}\right) \cdot L[0] \oplus L_*[j] \\ &= F_K\left(N||10^n - |N| - 1, 0^m\right) \oplus \gamma_{i-1} \cdot L[0] \oplus 2^{\texttt{ntz}(i)} \cdot L[0] \oplus L_*[j] \\ &= \Delta_K(N, i-1, j) \oplus 2^{\texttt{ntz}(i)} \cdot L[0] \\ &= \Delta_K(N, i-1, j) \oplus L[\texttt{ntz}(i)].\end{aligned}$$

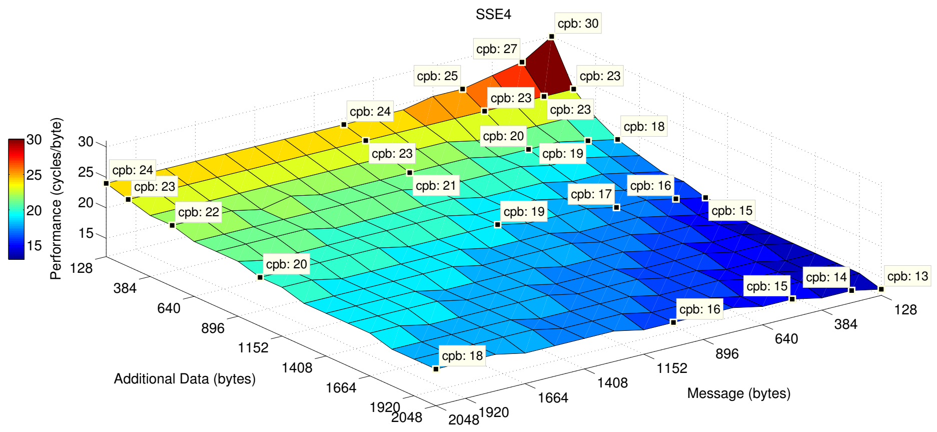


Figure A.5 – **3D surface plot of the performance of the OMD-sha512 SSE4 implementation** with increasing message and AD length. Graph from Ankele and Ankele [AA14].

Secondly, for any j, j' from the acceptable range we have

$$\begin{aligned}\Delta_K(N,i,j') &= F_K\left(N||10^{n}-|N|-1,0^{m}\right)\oplus\gamma_i\cdot L[0]\oplus L_*[j']\\ &= F_K\left(N||10^{n}-|N|-1,0^{m}\right)\oplus\gamma_i\cdot L[0]\oplus L_*[j]\oplus L_*[j]\oplus L_*[j']\\ &= \Delta_K(N,i,j)\oplus j\cdot L_*\oplus j'\cdot L_*\\ &= \Delta_K(N,i,j)\oplus(\langle j\rangle_n\oplus\langle j'\rangle_n)\cdot L_*\\ &= \Delta_K(N,i,j)\oplus L_*[\mathsf{str2num}\left(\langle j\rangle_n\oplus\langle j'\rangle_n\right)].\end{aligned}$$

These are the two rules we use in the specification of p-OMD in Section 5.4.

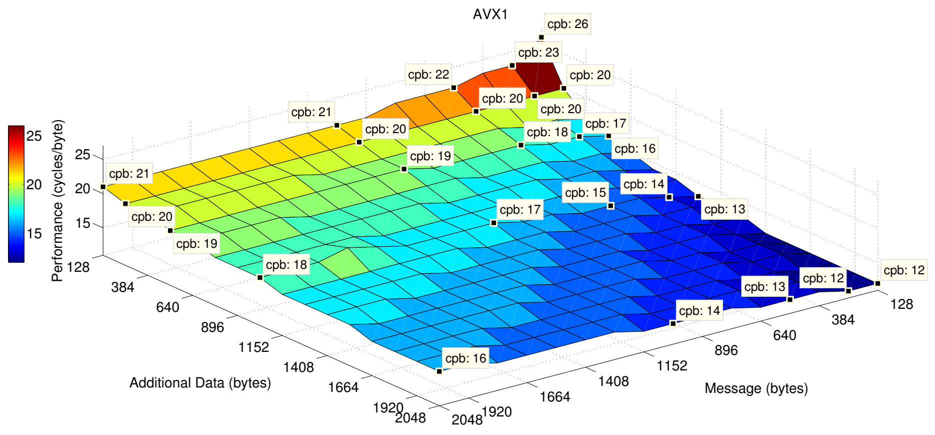


Figure A.6 – **3D surface plot of the performance of the OMD-sha512 AVX1 implementation** with increasing message and AD length. Graph from Ankele and Ankele [AA14].

Appendix B

Additional Notions, Separations and a Survey of OAE-like claims

B.1 Divergence of OAE1-like Claims

All that is reported in this section is based on what was published by early 2015.

A survey of the literature (published by 2015) shows increasingly strong rhetoric surrounding nonce-reuse security of online schemes. We document this trend. In doing so we identify some of the notions (all quite weak, in our view) that have come to be regarded as nonce-reuse misuse-resistant.

Shifting language The paper defining MRAE [RS06b] never suggested that nonce-reuse was OK; it said that an MRAE scheme must do "as well as possible with whatever IV is provided" [RS06b, p. 1]. Elaborating, the authors "aim for an AE scheme in which if the IV is a nonce then one achieves the usual notion for nonce-based AE; and if the IV does get repeated then authenticity remains and privacy is compromised only to the extent that [one reveals] if this plaintext is equal to a prior one, and even that ... only if both the message and its header have been used with this particular IV" [RS06b, p. 12–13].

The FFL paper indicates that the authors wish "to achieve both simultaneously: security against nonce-reusing adversaries ... and support for on-line-encryption" [FFL12, p. 197]. While the authors understood that they were weakening MRAE, they saw the weakening as relatively inconsequential: they say that their scheme, McOE, "because of being on-line, satisfies a *slightly weaker* security definition against nonce-reusing adversaries" [FFL12, p. 198] (emphasis ours). The paper did not investigate the definitional consequences of this weakening.

An early follow-on to FFL, the COPA paper, asserts that OAE1 schemes are distinguished by "not relying on the non-reuse of a nonce" [ABL+13, p. 438]. Andreeva et al. classify AE schemes according to the type of initialization vector (IV) one needs: either

OAE1 Leaks equality of block-aligned prefixes, formalized by comparing $\mathcal{E}_K$ with: a random n-bit-blocksize online permutation tweaked by the nonce, AD and plaintext; followed by a random τ-bit function of the nonce, AD, and plaintext. Schemes1: COPA [ABL$^+$14c], Deoxys [JNPS14a], Joltik [JNPS14b], KIASU [JNPS14c], Marble[Guo14b], McOE [FFL12], SHELL [Wan14], POET [AFF$^+$14a, AFF$^+$14b], Prøst-COPA [KLL$^+$14] Schemes2: ++AE [Rec14]
OAE1a Leaks equality of block-aligned prefixes, formalized by comparing $\mathcal{E}_K$ with: a random n-bit-blocksize online *function* tweaked by the nonce, AD and plaintext; followed by a random τ-bit function of the nonce, AD, and plaintext. Schemes1: APE[ABB$^+$14c], ELmD[DN14a], ELmE[DN14b], Prøst-APE[KLL$^+$14]
OAE1b Leaks equality of block-aligned prefixes, formalized by comparing $\mathcal{E}_K$ with: a random n-bit-blocksize online *function* tweaked by the nonce and plaintext (but *not* the AD); followed by a random τ-bit function of the nonce, AD, and plaintext. The relaxation enables a compliant scheme to process the plaintext before the AD is presented. However it also renders a compliant scheme vulnerable to CCA, CPSS, and NM attacks even if AD values are unique. Schemes1: COBRA[ALMY14, ABL$^+$14b]
OAE1c Leaks equality of any blocks at the same position. E.g., if ciphertexts C and C' arise from 4-block plaintexts $P = \mathrm{A}\Vert\mathrm{B}\Vert\mathrm{C}\Vert\mathrm{D}$ and $P' = \mathrm{E}\Vert\mathrm{B}\Vert\mathrm{F}\Vert\mathrm{D}$ then $C_2 = C'_2$ and $C_4 = C'_4$. Security is formalized by comparing $\mathcal{E}_K$ with: a function from n bits to n bits tweaked by the nonce and an integer, the position; followed by a random tag. Schemes1: Minalpher [STA$^+$14]
OAE1d Leaks equality of block-aligned prefixes and the XOR of the block directly following this prefix. E.g., if C, C' arise from 4-block plaintexts $P = \mathrm{A}\Vert\mathrm{B}\Vert\mathrm{C}\Vert\mathrm{D}$ and $P' = \mathrm{A}\Vert\mathrm{B}\Vert\mathrm{E}\Vert\mathrm{F}$ we always have $C_1 = C'_1$, $C_2 = C'_2$, and $C_3 \oplus C'_3 = \mathrm{C} \oplus \mathrm{E}$. Ciphertexts C, C' arising from 4-block plaintexts $P = \mathrm{A}\Vert\mathrm{B}\Vert\mathrm{C}\Vert\mathrm{D}$ and $P' = \mathrm{E}\Vert\mathrm{F}\Vert\mathrm{G}\Vert\mathrm{H}$ will have $C_1 \oplus C'_1 = \mathrm{A} \oplus \mathrm{E}$. Schemes2: Artemia [AAB14] CBEAM [Saa14], ICEPOLE [MGH$^+$14], iFeed [ZWHS14], Jambu [WH14b], Keyak [BDP$^+$14b], MORUS [WH14a], NORX [AJN14], STRIBOB [SB14]
NAE1 Retains full security as long as all (N, A) pairs are unique among the encryption queries. If a pair repeats, all privacy is lost, but authenticity remains unchanged. Schemes1: CLOC [IMG$^+$14a], SILC [IMG$^+$14b]
NAE0 Retains full security as long as all (N, A) pairs are unique among the encryption queries. If a pair repeats, all security is forfeit. Schemes1: NORX [AJN14], Trivia-ck [AC14] Schemes2: OTR [Min14b, Min14a]

Figure B.1 – **A menagerie of OAE notions and schemes.** All of the schemes are CAESAR submissions except ElmE and McOE. Schemes1 lists proposals that claim some flavor of nonce-reuse misuse resistance. Schemes2 lists proposals that didn't, yet are or were marked as such in the AE Zoo [AKL$^+$14] or AFL survey [AFL16].

random, *nonce*, or *arbitrary*. A scheme satisfying OAE1 is understood to be an arbitrary-IV scheme, where "no restrictions on the IV are imposed, thus an adversary may choose any IV for encryption" [ABL$^+$14a, p. 9]. The authors add that "Often a deterministic AE scheme does not even have an IV input" [ABL$^+$14a, p. 9]. The linguistic progression reaches its logical conclusion in the rebranding of OAE1-secure schemes as *nonce-free*, as seen, for example, in talks of Guo [Guo14a, slide 2] and Lauridsen [BLT14, Slides 4, 6].

We have thus seen a transformation in language, played out over eight years, taking us from a strong definition (MRAE) pitched as trying to capture the best one can do when a nonce gets reused to a comparatively weak definition (OAE1) nowadays pitched as being so strong so as to render nonces superfluous. Meanwhile, the best-one-can-do positioning of MRAE was mirrored in the online setting. The COPA authors indicate that their mode achieves "the maximum attainable for single pass schemes" [ABL$^+$14c, p. 7]. Identical language is found in the COBRA submission [ABL$^+$14b, p. 7]. In our view, such claims are wrong; there would seem to be a gap between OAE1 and OAE2 security, what we illustrate in Appendix B.3.

Weaker notions Concurrent with the rhetoric for what OAE1 delivers being ratcheted up, weakened variants of OAE1 have proliferated. We document this trend in Figure B.1, which introduces a variety of OAE notions. They are all weaker than OAE1 except for OAE1a; by standard arguments, OAE1 and OAE1a are quantitatively close if the blocksize is reasonably large. In this race to the bottom, it may seem as though the scheme comes first and whatever properties it provides is branded as *some* form misuse resistance.

The number of different OAE definitions, and their meanings, has never been clear. The evolution of what has been indicated in the Nonce-MR column in the AE Zoo website [AKL$^+$14] illustrates the struggle of researchers trying to accurately summarize the extent of nonce-reuse misuse-resistance for tens of AE schemes.[1] Our own attempt at sorting this out, Figure B.1, is not definitive. We do not formalize the notions in this table except for OAE1. (Some of the definitions are obvious, some are not.) The table is based on both author assertions (Schemes1) and assertions of others (Schemes2). The OAE1x notions only consider security for messages that are blocksize multiples.

B.2 MRAE Resists CPSS

We evidence that MRAE-secure schemes, unlike OAE1-secure schemes, resist CPSS attack. The MRAE notion is much stronger still, but a result like what we give is a starting point.

Suppose that the secret suffix S is generated by an efficient sampler $\mathcal{S}$. Let $\mathbf{Adv}_{\mathcal{S}}^{\mathbf{guess}}$ denote the min-entropy of the distribution generated by $\mathcal{S}$. Let $\mathscr{A}$ be a CPSS adversary attacking an AE scheme Π. Consider the following MRAE adversary $\mathscr{B}$ attacking Π. It generates the suffix $S \leftarrow\!\!\$\ \mathcal{S}$ and runs $\mathscr{A}$. For each prefix P that $\mathscr{A}$ produces, if $\mathscr{A}$ repeats a prior prefix then $\mathscr{B}$ gives the consistent answer. Otherwise, it queries $P \| S$ to its encryption oracle, and returns the answer to $\mathscr{A}$. If $\mathscr{A}$ can reproduce S then $\mathscr{B}$ outputs 1; otherwise it outputs 0. If $\mathscr{B}$'s oracle returns random strings ($\mathscr{B}$ is playing the game $\mathbf{mrae\text{-}I}_{\Pi}$) then $\mathscr{A}$ can guess S with probability at most $\mathbf{Adv}_{\mathcal{S}}^{\mathbf{guess}}$, since it only receives random answers independent of S. Hence the chance that $\mathscr{A}$ can guess S in the CPSS attack against Π is at most $\mathbf{Adv}_{\Pi}^{\mathbf{mrae}}(\mathscr{B}) + \mathbf{Adv}_{\mathcal{S}}^{\mathbf{guess}}$.

[1]We note that after the publication of the paper corresponding to this chapter, AE Zoo significantly updated their Nonce-MR column.

B.3 Separating OAE1$[n]$ and OAE2$[0, n]$

OAE1 is weaker than OAE2 in the sense that the former does not support arbitrary segmentation or demand security over arbitrary strings. This brief section argues a more specific claim: that OAE1 with blocksize and tagsize n remains weaker than OAE2 with a $(0, n)$-expanding scheme and all segments required to have exactly n bits. (To address the syntactic mismatch, we'll assume that every $\boldsymbol{A}$ satisfies $\boldsymbol{A}[i] = \varepsilon$ for every $1 < i \leq |\boldsymbol{A}|$, so that OAE2 can be viewed as operating on a single AD string, instead of an AD vector.) This is true even if we fix a reasonably large value of n, say $n = 128$. We ignore mundane matters of mismatched syntax that would have to be dealt with in a more formal treatment (i.e., that OAE1 and OAE2 schemes are very different *kinds* of objects).

So consider an OAE1-secure scheme $\Pi = (\mathbf{K}, \mathbf{E}, \mathbf{D})$ of blocksize n. Assume that all keys output by $\mathbf{K}$ also have n bits, which is the most common case for $n = 128$. Suppose that for scheme Π one can recover the m-th block of the (putative) plaintext from K, N, A, and the first m blocks of ciphertext, which again holds for typical schemes, like COPA [ALMY14] and McOE [FFL12]. Now consider the following scheme $\tilde{\Pi} = (\mathbf{K}, \tilde{\mathbf{E}}, \tilde{\mathbf{D}})$. For any $X \in (\{0,1\}^*)^n$, let $\tilde{\mathbf{E}}_K^{N,A}(K \| X) = \mathbf{E}_K^{N,A}(M_0 \| X)$, where M_0 is the first block of the putative plaintext obtained from decrypting 0^{2n} under key K with nonce N and AD A; and let $\tilde{\mathbf{E}}_K^{N,A}(M_0 \| X) = \mathbf{E}_K^{N,A}(K \| X)$. Let $\tilde{\mathbf{E}}_K^{N,A}$ coincide with $\mathbf{E}_K^{N,A}$ on all other points. Then the scheme $\tilde{\Pi}$ ought still to be OAE1-secure. For given an adversary $\mathscr{A}$ attacking $\tilde{\Pi}$, one can transform it to an equally efficient adversary $\mathscr{B}$ attacking Π that outputs 1 if the first block of some ciphertext is 0^n, and the probability that $\mathscr{A}$ can query some $M_0 \| X$ to the encryption oracle is at most $\mathbf{Adv}_{\Pi}^{\mathbf{oae1}}(\mathscr{B})$. But $\tilde{\Pi}$ is not OAE2-secure, as an adversary can query 0^{2n} to the decryption oracle to learn K. In brief, we can adjust an OAE1-secure scheme to fail miserably in the presence of a decryption capability, the adjustment irrelevant for OAE1 security.

We emphasize that the above counterexample does not imply that common OAE1-secure schemes with expansion τ fail to be OAE2-secure with expansion $(0, \tau)$ once reconceptualized and restricted. Such a determination would have to be made on a case-by-case basis, asking if supplementing the adversary's capabilities with an online decryption oracle would violate indistinguishability. We haven't carried out such investigations because even when an OAE1 scheme *is* OAE2-secure once restricted and reconceptualized, we are not suggesting this would make it a desirable way to address online-AE: in particular, expansion parameters of $(0, \tau)$ are likely to be a poor choice in most settings, since they provide no authenticity assurance until a ciphertext's end. This is why our focus has been on τ-expanding schemes, where all segments are afforded the same authenticity guarantees. It also seems undesirable to insist on segmenting messages along n-bit boundaries for some small, fixed n, and to fail to define security for messages that are not blocksize multiples.

B.4 IND-CCA and RAE Security of AE Schemes

```
proc initialize                    ind-cca-R_Π
$K \leftarrow_\$ \mathcal{K}$
$\mathcal{V} \leftarrow \varnothing, \mathcal{X} \leftarrow \varnothing, \mathcal{Y} \leftarrow \varnothing$

oracle Enc$(N, A, M)$
if $N \in \mathcal{X}$ then return $\perp$
if $(N, A, M) \in \mathcal{V}$ then return $\perp$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{N\}$
$C \leftarrow \mathcal{E}(K, N, A, M)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(N, A, C)\}$
return $C$

oracle Dec$(N, A, C)$
if $(N, A, C) \in \mathcal{Y}$ then return $\perp$
$M \leftarrow \mathcal{D}(K, N, A, \tau, C)$
if $M \neq \perp$
  $\mathcal{V} \leftarrow \mathcal{V} \cup \{(N, A, M)\}$
return $M$
```

```
proc initialize                    ind-cca-I_Π
$K \leftarrow_\$ \mathcal{K}$
$\mathcal{V} \leftarrow \varnothing, \mathcal{X} \leftarrow \varnothing, \mathcal{Y} \leftarrow \varnothing$

oracle Enc$(N, A, M)$
if $N \in \mathcal{X}$ then return $\perp$
if $(N, A, M) \in \mathcal{V}$ then return $\perp$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{N\}$
$C \leftarrow_\$ \{0,1\}^{|M|+\tau}$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(N, A, C)\}$
return $C$

oracle Dec$(N, A, C)$
if $(N, A, C) \in \mathcal{Y}$ then return $\perp$
$M \leftarrow \mathcal{D}(K, N, A, \tau, C)$
if $M \neq \perp$
        $\mathcal{V} \leftarrow \mathcal{V} \cup \{(N, A, M)\}$
return $M$
```

Figure B.2 – **IND-CCA security.** Games for defining IND-CCA security of a nonce-based AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with stretch τ.

Here we define the IND-CCA security of the nonce-based AE schemes (defined in Section 2.4) and RAE [HKR15] security of variable-stretch schemes (defined in Section 8.5).

Definition B.1 (IND-CCA security). *Given a nonce-based AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with ciphertext expansion τ and an adversary $\mathscr{A}$, we define the advantage of $\mathscr{A}$ in breaking the IND-CCA security of Π in a chosen ciphertext attack (with help of the games* **ind-cca-*R*** *and* **ind-cca-*I*** *in Figure B.2) as*

$$\mathbf{Adv}_{\Pi}^{\textbf{ind-cca}}(\mathscr{A}) = \Pr\left[\mathscr{A}^{\textbf{ind-cca-}\boldsymbol{R}_\Pi} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\textbf{ind-cca-}\boldsymbol{I}_\Pi} \Rightarrow 1\right].$$

If $\mathbf{Adv}_{\Pi}^{\textbf{ind-cca}}(\mathscr{A}) \leq \epsilon$ for all adversaries $\mathscr{A}$ run in time t, and make no more than q_e encryption queries and no more than q_d decryption queries such that their total data complexity is no more than σ bits, then we say that Π is $(\epsilon, t, q_e, q_d, \sigma)$-IND-CCA secure.

Definition B.2 (RAE security). *Given a variable-stretch AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ and an adversary $\mathscr{A}$, we define the advantage of $\mathscr{A}$ in breaking the robust-AE security of Π in a chosen ciphertext attack (with help of the games* **rae-*R*** *and* **rae-*I*** *in Figure B.2) as*

$$\mathbf{Adv}_{\Pi}^{\textbf{rae}}(\mathscr{A}) = \Pr[\mathscr{A}^{\textbf{rae-}\boldsymbol{R}_\Pi} \Rightarrow 1] - \Pr[\mathscr{A}^{\textbf{rae-}\boldsymbol{I}_\Pi} \Rightarrow 1].$$

If $\mathbf{Adv}_{\Pi}^{\mathbf{rae}}(\mathscr{A}) \leq \epsilon$ *for all adversaries* $\mathscr{A}$ *run in time* t*, and make no more than* q_e *encryption queries and no more than* q_d *decryption queries such that their total data complexity is no more than* σ *bits, then we say that* Π *is a* $(\epsilon, t, q_e, q_d, \sigma)$*-secure robust AE scheme.*

```
proc initialize                                  $\boxed{\textbf{rae-R}_{\Pi}}$
$K \leftarrow\$ \mathcal{K}$

proc $\mathrm{Enc}(N, A, \tau, M)$
return $\mathcal{E}(K, N, A, \tau, M)$

proc $\mathrm{Dec}(N, A, \tau, C)$
return $\mathcal{D}(K, N, A, \tau, C)$
```

```
proc initialize                                  $\boxed{\textbf{rae-I}_{\Pi}}$
for $N, A, \tau \in \mathcal{N} \times \{0,1\}^* \times \mathbb{N}$ do
    $\pi_{N,A,\tau} \leftarrow\$ \mathrm{Inj}(\tau)$

proc $\mathrm{Enc}(N, A, \tau, M)$
return $\pi_{N,A,\tau}(M)$

proc $\mathrm{Dec}(N, A, \tau, C)$
if $\exists M \in \{0,1\}^*$ s.t. $\pi_{N,A,\tau}(M) = C$ then
    return $M$
return $\perp$
```

Figure B.3 – **RAE security.** Defining security for a robust AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with nonce space $\mathcal{N}$. $\mathrm{Inj}(()\tau)$ denotes the set of all injective, τ-expanding functions from $\{0,1\}^*$ to $\{0,1\}^{\geq \tau}$.

Appendix C

Descriptions of 3$^{\text{rd}}$ Round CAESAR Candidates

In this appendix, we briefly outline CCM, GCM and the 3$^{\text{rd}}$ round CAESAR candidates.

C.1 AES-CCM

CCM [WHF03b] combines encrypted CBC MAC for authentication with CTR mode for message encryption. We assume the blockcipher E is AES. To encrypt a query (N, A, M), CCM first computes the value $U = \text{CBCMAC}_K(B)$ where the string B is an injective, prefix-free encoding of N, A, M, τ and other parameters, s.t. 128 divides $|B|$. In particular, N and $|M|$ is encoded in B_0 and $|A|$ is encoded in B_1. The ciphertext is computed as $C_i = M_i \oplus E_K(\mathsf{I}(N, i))$ for $i = 1, \ldots, |M|_{128}$ where $\mathsf{I}(N, i)$ is a 128-bit injective encoding of N and i, such that $\mathsf{I}(N, i) \neq B_0$ for $i \geq 0$. The tag is computed as $T = \mathsf{left}_\tau\left(U \oplus E_K(\mathsf{I}(N, 0))\right)$.

C.2 AES-GCM

AES-GCM [MV04] combines counter mode for message encryption with a Wegman-Carter MAC (based on a polynomial AXU hash called GHASH) for authentication. It uses AES as the blockcipher E, and derives a key for GHASH as $L = E_K(0)$. GHASH takes two strings as input and computes

$$\text{GHASH}_L(A, C) = \bigoplus_{i=1}^{\ell} L^{\ell-i+1} \cdot X_i, \quad \text{with} \quad X = A\|0^*\|C\|0^*\|\langle|A|\rangle_{64}\|\langle|C|\rangle_{64},$$

where $\ell = |A|_{128} + |C|_{128} + 1$ and the multiplications are done in $\mathsf{GF}(2^{128})$. To encrypt a query (N, A, M), we first set $I \leftarrow N\|0^{31}1$ if $|N| = 96$ and to $I \leftarrow \text{GHASH}_L(\varepsilon, N)$ if $|N| \neq 96$. Then we compute the ciphertext C as counter mode encryption of M, using[1]

[1]We abuse the notation slightly; the incrementation is done with integer representation of $\mathsf{right}_{32}(Y)$.

$\mathsf{inc}(Y) \mapsto \mathsf{left}_{96}(Y) \| (\mathsf{right}_{32}(Y) + 1)$ as the incrementation function and $\mathsf{inc}(I)$ as the initial counter value. Then we compute the tag as $T = \mathsf{left}_\tau(\mathrm{GHASH}_L(A, C) \oplus E_K(I))$.

C.3 AEZ v5

AEZ encryption [HKR17] (in the AEZ-core case, i.e. the general one for $|M| \geq 256$) has the following structure

$$\mathsf{Encrypt}(K, N, A, \tau, M) = f(I, J, L, \Delta, M),$$

where $(I, J, L) = \mathsf{KDF}(K)$ and the value $\Delta = H(I, J, L, \tau, N, A)$ is computed using a dedicated hash function H. So, getting (I, J, L) is equivalent to getting K in terms of key recovery attacks. All binary inputs can have an arbitrary length and are internally processed in blocks of 128 bits.

We note that, whenever we encrypt the same message M with the same key K, any collision on the hash function necessarily results in a collision on the ciphertext. So, it is easy to detect collisions on the hidden variable Δ in a chosen message attack.

In the context of AEZ, we denote by $k = (0, J, I, L, 0)$ a sequence of 5 blocks which are used as round keys in $\mathsf{AES4}$, a subroutine of H based on AES [DR02] reduced to 4 rounds. We further denote $H(I, J, L, \tau, N, A) = h_k(\tau, N, A)$. We will use another subroutine $E_K^{j,i}$ defined by

$$E_K^{j,i}(X) = \mathsf{AES4}_k(X \oplus jJ \oplus 2^{\lceil i/8 \rceil} I \oplus (i \bmod 8)L)$$

for $j \geq 0$ and where integer-block multiplication denotes the classical $\mathsf{GF}(2^{128})$ multiplication.

C.4 OCB (OCB v1.1)

OCB [KR16] (a.k.a. OCB3) uses AES as the blockcipher E and derives two secret offset values: L from the secret key K only, and R from K, the nonce N (a string of no more than 120 bits), and the stretch τ. Each plaintext block M_i is encrypted into C_i by

$$C_i = E_K(M_i \oplus \Delta_i) \oplus \Delta_i \quad \text{with} \quad \Delta_i = R \oplus \gamma_i \cdot L,$$

where $\cdot$ denotes the multiplication in $\mathsf{GF}(2^{128})$ and γ_i is a 128-bit block that takes a unique value for every $1 \leq i \leq 2^{120}$. The nonce N has up to 120 bits. We have that $L = 4 \cdot E_K(0)$. The way to compute R is a bit complicated but there is a simple particular case: when the 6 least significant bits of N are all zero, then $R = E_K(0^*1 \| N)$.

When the last blocks M_ℓ of M and A_a of A (with $m = |M|_{128}$ and $a = |A|_{128}$) are complete and $\tau = 128$, the tag is computed as

$$T = E_K\left(2^{-1} \cdot L \oplus \Delta_\ell \oplus \bigoplus_{i=1}^{\ell} M_i\right) \oplus H_K(A) \text{ where } H_K(A) = \bigoplus_i E_K(A_i \oplus \gamma_i \cdot L)$$

and $\cdot$ denotes the multiplication in $\mathsf{GF}(2^{128})$. The attacks we describe in Section 9.9 can be easily generalized to the case when the last block of M and the last block of A are not 128 bits long.

C.5 AES-OTR v3.1

AES-OTR v3.1 with parallel AD processing [Min16] produces a tag $T = \mathsf{left}_\tau(\mathsf{TA} \oplus \mathsf{TE})$ where TA and TE are partial tags for A and the pair (N, M), respectively. We assume that $|N| = 120$. Note that TA does not depend on the nonce N. When $|A|$ is a multiple of 128, it is computed as

$$\mathsf{TA} = E_K\left(\left(\bigoplus_{i=1}^{a-1} E_K(A_i \oplus 2^i \cdot Q)\right) \oplus A_a \oplus 2^{a-1} \cdot 3^3 \cdot Q\right)$$

where $a = |A|_{128}$, E_K is AES with key K, $Q = E_K(0)$ and $\cdot$ denotes the multiplication in $\mathsf{GF}(2^{128})$. When $|M|_{128} = 2\ell$ and $|M_{2\ell}| = 128$, blocks are encrypted in pairs (M_{2i-1}, M_{2i}) into (C_{2i-1}, C_{2i}) by a two-round Feistel scheme

$$C_{2i-1} = E_K(2^{i-1} \cdot L \oplus M_{2i-1}) \oplus M_{2i} \qquad C_{2i} = E_K(2^{i-1} \cdot 3 \cdot L \oplus C_{2i-1}) \oplus M_{2i-1}$$

where $L = E_K(\epsilon(\tau)\|0^*1\|N)$ and $\epsilon(\tau)$ is a 7-bit encoding of τ. The tag TE is obtained by

$$\mathsf{TE} = E_K\left(7 \cdot 2^{\ell-1} \cdot 3 \cdot L \oplus \bigoplus_{i=1}^{\ell} M_{2i}\right).$$

C.6 CLOC and SILC

CLOC and SILC v3 [IMG+16] use nonces of 96 bits with AES as the blockcipher E. They compute $V = \mathsf{HASH}_K(N, A)$, then $C = \mathsf{ENC}_K(V, M)$, and finally $T = \mathsf{PRF}_K(V, C)$. In ENC, we compute $C_1 = M_1 \oplus E_K(V)$. Then, $C_2, \ldots, C_m$ is a function of K, C_1, and $M_2, \ldots, M_m$ only (where $m = |M|_{128}$). More precisely, we have

$$C_i = M_i \oplus E_K(\mathsf{fix1}(C_{i-1}))$$

for $i > 0$, where $\mathsf{fix1}$ just forces the most significant bit to 1.

C.7 Deoxys v1.41

Deoxys v1.41 [JNP16] has two instantiations: Deoxys-I, which claims no nonce-misuse security, and Deoxys-II, which claims to resist nonce misuse attacks. The internal design of Deoxys-I is very similar to OCB, except that it relies on an ad-hoc tweakable encryption E_K^T instead of AES with the input and output masks Δ. The block size of E is 128 bits. In both instances we have

$$H_K(A) = \bigoplus_i E_K^{2\|(i-1)}(A_i)$$

(when the last block of A is complete) which is nonce-independent. In Deoxys-I, we have

$$T = E_K^{1\|N\|\ell}\left(\bigoplus_{i=1}^{\ell} M_i\right) \oplus H_K(A) \text{ and } C_i = E_K^{0\|N\|(i-1)}$$

(when the last block M_ℓ of M is complete). In Deoxys-II, we have (when the last block M_ℓ of M is complete)

$$T = E_K^{1\|0^4\|N}\left(H_K(A) \oplus \bigoplus_{i=1}^{\ell} E_K^{0\|(i-1)}(M_i)\right) \text{ and} C_i = M_i \oplus E_K^{1\|(T\oplus(i-1))}(0^8\|N).$$

C.8 Tiaoxin-346

Tiaoxin-346v2.1 [Nik16] loads the key K and the nonce N in a state, then applies a reversible transformation. The state consists of three arrays $T[3]$, $T[4]$, and $T[6]$, of respectively 3, 4, and 6 blocks. Each block is represented by a 4×4 matrix of bytes. Given a (K, N-initialized) state $(T[3], T[4], T[6])$, we load the plaintext by pairs (M_{2i}, M_{2i+1}) of blocks and output two blocks (C_{2i}, C_{2i+1}) for $i = 0, \ldots, |M|_{2\cdot 128}$. This operation works in five steps:

1. $T[3] \leftarrow R(T[3], M_{2i})$,
2. $T[4] \leftarrow R(T[4], M_{2i+1})$,
3. $T[6] \leftarrow R(T[6], M_{2i} \oplus M_{2i+1})$,
4. $C_0 \leftarrow T[3]_0 \oplus T[3]_2 \oplus T[4]_1 \oplus (T[6]_3 \& T[4]_3)$,
5. $C_1 \leftarrow T[6]_0 \oplus T[4]_2 \oplus T[3]_1 \oplus (T[6]_5 \& T[3]_2)$.

The $R(T[s], M)$ operation consists of

$$R(T[s], M) = (A(T[s]_{s-1}) \oplus T[s]_0 \oplus M, A(T[s]_0) \oplus Z_0, T[s]_1, \ldots, T[s]_{s-2})$$

where Z_0 is a constant and A is one AES round without round key:

$$A(x) = \mathsf{MixColumns}(\mathsf{ShiftRows}(\mathsf{SubBytes}(x))).$$

C.9 AEGIS v1.1

AEGIS v1.1 [WP16] uses the notion of *state*. In the AEGIS-128 version (the lightest of the three proposed ones), one state consists of five AES states, i.e. five 4×4 matrices of bytes. AEGIS first computes an initial state which depends on the key K and the nonce N. Then, neither K nor N is used any more. It processes A and M as a sequence of 4×4 matrices of bytes. Each matrix X is processed to update the state $S_0, \ldots, S_4$ into

$$\begin{pmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \end{pmatrix} = \begin{pmatrix} R(S_4) \oplus X \oplus S_0 \\ R(S_0) \oplus S_1 \\ R(S_1) \oplus S_2 \\ R(S_2) \oplus S_3 \\ R(S_3) \oplus S_4 \end{pmatrix}$$

where R is a single AES round function without the addition of a round key. Before this transformation, $X \oplus S_1 \oplus (S_2 \& S_3) \oplus S_4$ is revealed for encryption, if X is a message block. After all blocks are processed, a function transforms the state using the length of A and M, and a tag is extracted from the result.

C.10 ACORN v3

ACORN v3 [Wu16] uses the notion of *state* and processes the bits of A and M iteratively, i.e. the "block" size is 1 bit. ACORN-128 has a state S of 293 bits. First, ACORN initializes a state depending on the key K and the nonce N. Then, it processes each bit of A iteratively by

$$S_{i+1} = \mathsf{StateUpdate128}(S_i, m_i, \mathsf{ca}_i, \mathsf{cb}_i)$$

where m_i is a new bit of A, ca_i is a control bit set to 1, and cb_i is a control bit set to 1. After processing A, the process continues for 128 more iterations with m_i set to 1 in the first iteration and to 0 in the remaining 127 iterations. It continues again with $m_i = 0$ and $\mathsf{ca}_i = 0$ for 128 more iterations. Encryption can then start with the same iterative process, where m_i is a new bit of M, then set to 1 once, then set to 0 255 times. One difference is that cb_i is set to 0. The other difference is that there is one output bit produced for encryption per state update when bits of the message are processed. Namely, we have

$$\begin{aligned}
o &= |A| + 256\\
S_{i+1+o} &= \mathsf{StateUpdate128}(S_{i+o}, M_i, 1, 0)\\
C_i &= M_i \oplus \mathsf{ks}_{i+o}\\
\mathsf{ks}_{i+o} &= S_{i+o,12} \oplus S_{i+o,154} \oplus \mathsf{maj}(S_{i+o,235}, S_{i+o,61}, S_{i+o,193})\\
&\quad \oplus \mathsf{ch}(S_{i+o,230}, S_{i+o,111}, S_{i+o,66})
\end{aligned}$$

where o is an offset, and maj and ch are two boolean functions of algebraic degree two. After processing the message bits, there are two sets of 128 iterations like for processing A. Then, there are 768 more iterations with various control bits and $m_i = 0$ before a tag is computed from the state. Another observation from the specifications shows that $S_{i+1,[0\cdots j]}$ is a linear function of $S_{i,[0\cdots j+1]}$ for $j < 292$ and that the last bit $S_{i+1,292} \oplus m_i \oplus \mathsf{maj}(S_{i+o,244}, S_{i+o,23}, S_{i+o,160})$ is also linear in S_i. So, $S_{i+j,[0\cdots k]}$ is a linear function of $S_{i,[0\cdots k+j]}$ for $k + j \leq 292$.

C.11 Ketje

Ketje v2 [BDP+16a] is a sponge-based mode for an iterated cryptographic permutation $f : \{0,1\}^b \to \{0,1\}^b$ with a tunable number of rounds, aggressively optimized for low computational cost. We focus on the main recommendation Ketje Sr (further simply Ketje) with 400-bit permutation, but the observations are easily generalised to the three remaining named instances.

Ketje operates over byte strings, and works with a rate $r = 36$ bits, capacity $c = 364$ bits, and tags of 64 bits. We assume the use of secret key and nonces of 128 bits.In an encryption query (K, N, A, M), Ketje first sets the state to the value $\text{Init}(K, N) = \langle 32\rangle_8 \| K \| 10^{119} \| N \| 10^*1$ and applies $n_{\text{start}} = 12$ rounds of the permutation f. Both the message M and AD A are partitioned into blocks of $r - 4$ bits, the last block of each possibly being shorter. The blocks are then processed using $n_{\text{step}} = 1$-round calls to f, as illustrated in Figure C.1, treated with two-bit domain separation flags and 10*1 padding. The tag is derived using two calls to f, first of which uses $n_{\text{stride}} = 6$ rounds.

C.12 Morus

Morus 640 v2 [WH16a] loads the key K and the nonce IV in a state and does a non-invertible initialization. The state consists of five 128-bit blocks $S_0, \ldots, S_4$ (for Morus, we start indexing blocks at 0). Then, the plaintext blocks M_i are processed iteratively

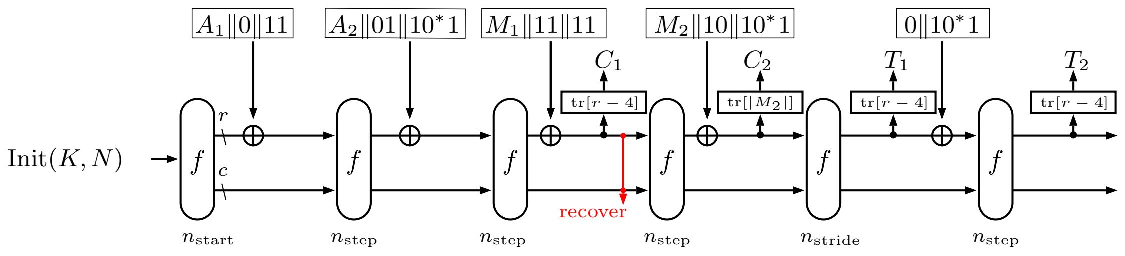


Figure C.1 – The encryption algorithm of Ketje [BDP+16a] with two-block A and two-block M as input. We let tr$[x]$ denote truncation to x leftmost bits.

in two steps:

1. $C_i = M_i \oplus S_0 \oplus (S_1 \lll 96) \oplus (S_2 \& S_3)$,
2. $(S_0, \ldots, S_4) \leftarrow \mathsf{StateUpdate}(S_0, \ldots, S_4, M_i)$.

The StateUpdate function is illustrated in Figure C.2.

C.13 NORX v3.0

NORX [AJN16] is a sponge-based mode which computes a "state" (R, S) from the secret key, the nonce N, and the parameters, then follows the sponge structure to absorb the associated data A, the message (at the same time it produces the ciphertext), the trailer data, then finally uses the key again to produce the tag. When processing one block of message M_i, NORX replaces (R, S) by a new state (C, S) with $C = M_i \oplus R$. We focus on an instance of NORX with a state size of 512 bits, with $|R| = 384$ and $|S| = 128$. The encryption algorithm of NORX is illustrated in Figure C.3.

C.14 Ascon

Ascon-128 v1.2 [DEMS16] is a sponge-based mode for an iterated cryptographic permutation $p : \{0,1\}^{320} \rightarrow \{0,1\}^{320}$ with tunable number of rounds (denoted as p^a for initialization and tag generation and p^b for the rest of the processing). Ascon works over a state S of 320 bits. We denote the outer (or the rate) part of the state S_r and the inner (or the capacity part) as S_c with $|S_r| = r$, $|S_c| = c$ and $S = S_r \| S_c$ (so $r + c = 320$). The keys, tags and nonces of Ascon are 128 bits long. We focus on Ascon128, with $r = 64$, $c = 256$, $a = 12$ and $b = 6$

When processing an encryption query (K, N, A, M), the associated data and message are both padded with 10^* padding so that $|M\|10^*|$ and $|A\|10^*|$ are both a multiple of r. The encryption algorithm is illustrated in Figure C.4.

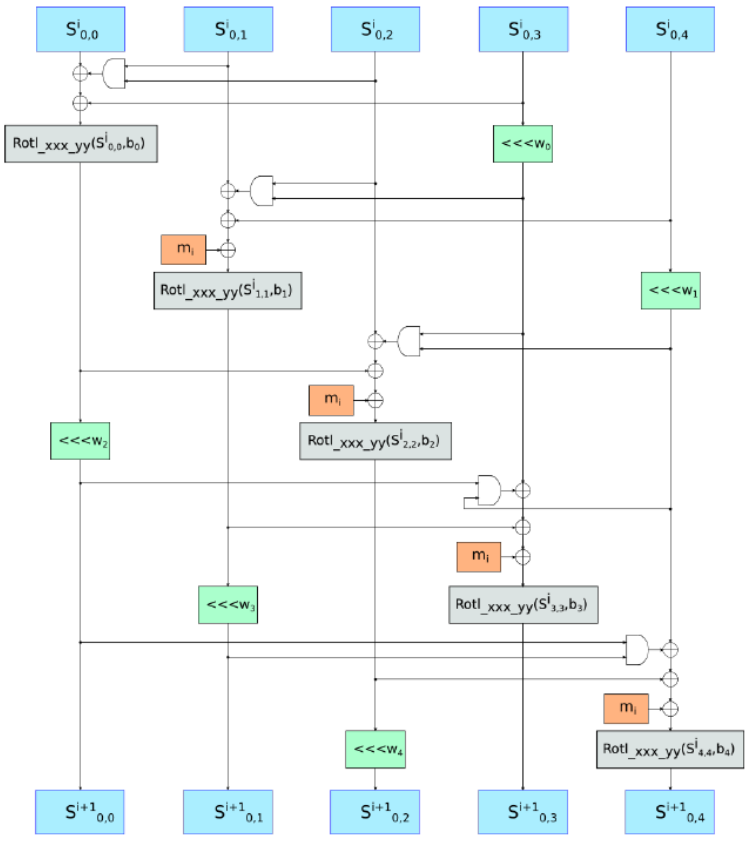


Figure C.2 – The StateUpdate function of Morus [WH16a].

C.15 Keyak

Keyak v2.2 [BDP+16b] is a sponge-based mode for an iterated cryptographic permutation $f : \{0,1\}^b \to \{0,1\}^b$. Keyak allows to tune many parameterssuch as degree of parallelism, a result of which the general description of Keyak is rather complicated and layered. We therefore focus on the main recommendation Lake Keyak with 1600-bit permutation and no parallelism (further simply Keyak).

Keyak operates over byte strings. It works with a state of $b = 1600$ bits, with capacity $c = 256$ bits, and tags of 128 bits. Keyak uses the whole state (including the inner part) to absorb data, working with absorption rate $R_a = 1536$ bits and squeezing rate

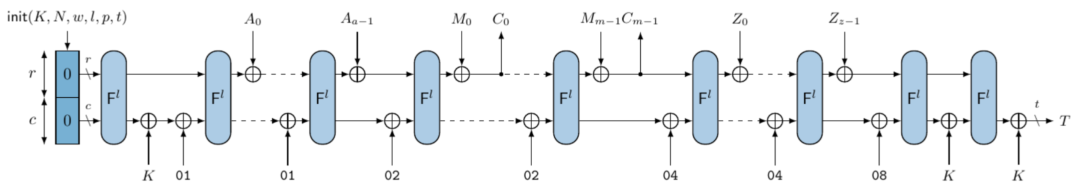


Figure C.3 – The encryption algorithm of NORX [AJN16].

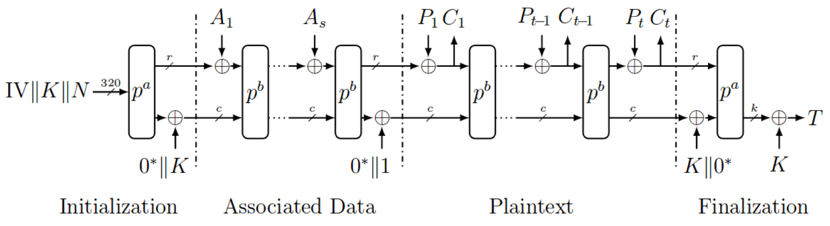


Figure C.4 – The encryption algorithm of Ascon [DEMS16]. Here P is the plaintext.

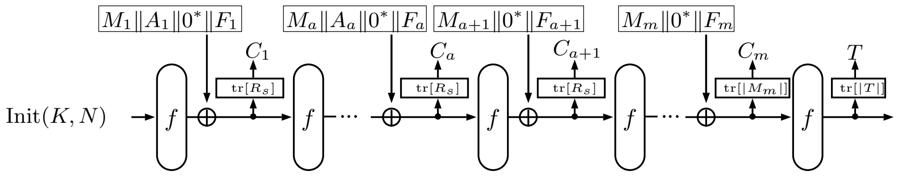


Figure C.5 – The encryption algorithm of Keyak [BDP+16b] when A is processed before M is. We let $\mathrm{tr}[x]$ denote truncation to x leftmost (outer) bits.

$R_s = 1344$ bits, which say how many bits can be absorbed and used (for encryption or tag) per call to f, respectively. We assume the use of nonces of 1200 bits and a secret key of 128 bits.

In an encryption query (K, N, A, M), Keyak first initializes the state to the value $\mathrm{Init}(K, N) = \langle 40\rangle_8\|K\|10^{183}\|N\|\langle 1\rangle_8\|\langle 0\rangle_8\|F_0$ and applies the permutation f, where F_0 is a 32-bit flag for domain separation. The message M is partitioned into blocks $M = M_1\|\ldots\|M_m$ of R_s bits (the last block possibly being shorter) and the AD is partitioned into blocks $A = A_1\|\ldots\|A_a\|\hat{A}_1\|\ldots\|\hat{A}_{\hat{a}}$ such that $|A_i| = (R_a - R_s)$ for $i = 0, \ldots, a-1$, $|A_a| \leq (R_s - R_a)$, and the $\hat{A}$ part can only be non-empty if $a = m$ and $|A_a| = (R_a - R_s)$. Depending of the length of M and A, three cases can occur; we illustrate the case when all bits of A are processed before the entire M is processed in Figure C.5. The flags F_i are 32 bit strings that encode (a) if the next evaluation of f produces a tag and the length of the tag, (b) the offset at which plaintext bytes end, (c) the offset at which AD bytes start and (d) the offset at which the AD bytes end. These flags ensure proper domain separation for all possible inputs.

C.16 COLM v1

COLM [ABD+16] first derives a secret L from the secret key L by $L = E_K(0)$, where E is AES [DR02]. Again, we use the notation in which the integer-by-block product is the $\mathsf{GF}(2^{128})$ multiplication. The COLM encryption takes a 64-bit nonce N, encodes some parameters param into 64 bits (these include the tag length), some associated data A and a plaintext M to produce a ciphertext C. Here, we restrict to A's and M's being sequences of full-length blocks, although COLM allows more flexibility in lengths. There is normally a padding scheme to transform a message $M_1, \ldots, M_{\ell-1}, M_\ell^*$ into the sequence $M_1, \ldots, M_{\ell+1}$, but it will play no role in the attack. We only have to keep in mind that it appends an additional block $M_{\ell+1}$ which is equal to the last one M_ℓ. We let $a = |A|_{128}$ and $\ell = |M|_{128}$ (the redundant block $M_{\ell+1}$ is appended by the encryption algorithm).

First, COLM computes sequences AA and MM as

$$\begin{aligned} \mathsf{AA}_0 &= (N \| \mathsf{param}) \oplus 3 \cdot L, & \mathsf{MM}_i &= M_i \oplus 2^i \cdot L \quad (i = 1, \ldots, \ell - 1), \\ \mathsf{AA}_i &= A_i \oplus 3 \cdot 2^i \cdot L \quad (i = 1, \ldots, a), & \mathsf{MM}_\ell &= M_\ell \oplus 7 \cdot 2^{\ell-1} \cdot L, \\ & & \mathsf{MM}_{\ell+1} &= M_{\ell+1} \oplus 7 \cdot 2^\ell \cdot L. \end{aligned}$$

Then we compute $Z_i = E_K(\mathsf{AA}_i)$ for $i = 0, \ldots, a$ and $X_i = E_K(\mathsf{MM}_i)$ for $i = 1, \ldots, \ell+1$. Then, $\mathsf{IV} = Z_0 \oplus \cdots \oplus Z_a$. There is a function ρ mapping a chaining value st and an input x to a chaining value st' and an output y defined by $\mathsf{st}' = x \oplus 2 \cdot \mathsf{st}$ and $y = x \oplus 3 \cdot \mathsf{st}$. We use IV as an initial chaining value and transform the sequence X iteratively with ρ to produce the sequence Y. Then, $\mathsf{CC}_i = E_K(Y_i)$ for $i = 1, \ldots, \ell + 1$, and finally,

$$\begin{aligned} C_i &= \mathsf{CC}_i \oplus 3^2 \cdot 2^i \cdot L \quad (i = 1, \ldots, \ell - 1), & C_\ell &= \mathsf{CC}_\ell \oplus 3^2 \cdot 7 \cdot 2^{\ell-1} \cdot L, \\ & & C_{\ell+1} &= \mathsf{CC}_{\ell+1} \oplus 3^2 \cdot 7 \cdot 2^\ell \cdot L. \end{aligned}$$

C.17 Jambu

Jambu v2.1 [WH16b] instantiated with the blockcipher AES works over $\nu = 64$-bit blocks. It works with a state of 3ν bits, that is iteratively updated with a function $F : \{0,1\}^k \times (\{0,1\}^\nu)^5 \to (\{0,1\}^\nu)^3$ which maps a secret key K, a state $R\|U\|V$, a domain separation constant γ and a message block M to an updated state $R'\|U'\|V' = F_K(R\|U\|V, \gamma, M)$, and which internally uses AES. The nonces in Jambu are 64 bits long.

Jambu first uses the nonce to initialize the state $R_0\|U_0\|V_0 \leftarrow F_K(0^\nu\|0^\nu\|N, 5, 0^\nu)$. Then the *always padded* (with 10^* padding) AD is processed in ν-bit blocks, computing $R_i\|U_i\|V_i \leftarrow F_K(R_{i-1}\|U_{i-1}\|V_{i-1}, 1, A_i)$ for $i = 1, \ldots, a$ with $|A\|10^*| = a\nu$.

The plaintext is partitioned in ν-bit blocks and encrypted by letting $R_{a+i}\|U_{a+i}\|V_{a+i} \leftarrow F_K(R_{a+i-1}\|U_{a+i-1}\|V_{a+i-1}, 0, M_i)$ and $C_i \leftarrow (M_i \oplus V_{a+i})$ for $i = 1, \ldots, m-1$ with $m = \lfloor |M|/\nu \rfloor + 1$. Then the final block of plaintext M_m is processed (if $|M| \bmod \nu = 0$, we set $M_m = \varepsilon$) by $R_{a+m}\|U_{a+m}\|V_{a+m} \leftarrow F_K(R_{a+m-1}\|U_{a+m-1}\|V_{a+m-1}, 0, M_m\|10^*)$ and $C_m \leftarrow \mathrm{trunc}_{|M_m|}(M_m\|10^* \oplus V_{a+m})$.

Finally we compute $R_{a+m+1}\|U_{a+m+1}\|V_{a+m+1} \leftarrow F_K(R_{a+m}\|U_{a+m}\|V_{a+m}, 3, 0^\nu)$, and the tag as $T \leftarrow R_{a+m+1} \oplus T_1 \oplus T_2$ with $T_1\|T_2 \leftarrow E_K(U_{a+m+1}\|V_{a+m+1})$.

Appendix D

Curriculum Vitae

Name:	Damian Vizár
E-mail:	damian.vizar@gmail.com
Citizenship:	Slovak

Education

2013-2018	**PhD Candidate in in Computer, Communication and Information Sciences** Area: Cryptography Supervised by Prof. Serge Vaudenay Ecole Polytechnique Fédérale de Lausanne, Switzerland
2011-2013	**Master degree in Applied Informatics** Slovak University of Technology, Slovakia
2008-2011	**Bachelor degree in Applied Informatics** Slovak University of Technology, Slovakia

Experience

2018	**Program committee member of CECC 2018**
2018	**Program committee member of FSE 2019**
2017	**Replacement teaching: Cryptography and Security**
Sep 2016	**Invited talk at ASK 2016**
Jun 2016	**Invited talk at CECC 2016**
May 2016	**Program committee member of SAC 2016**
Feb 2014 - Jun 2016	**A successful project within the Microsoft Research Joint Research Centre collaboration with EPFL**
Sep 2013 - Jan 2018	**Teaching assistant** (courses: Cryptography and Security course (4 semesters), Advanced Cryptography, Student seminar on Security Protocols and Applications, and Mathematical Analysis II)
Sep 2013 - Jan 2018	**Supervised 10 successful Master and Bachelor semester projects**
Mar-Jul 2012	**Erasmus Exchange** Ruhr Universität Bochum, Germany
Oct 2012- Jul 2013	**IT Support** ING Insurance Company, Slovakia

Honours

2016	**Awarded a Google PhD Fellowship**
2015	**Awarded the Outstanding teaching assistant award** from the School of computer and communication science Ecole Polytechnique Fédérale de Lausanne
2013	**Awarded a PhD fellowship** Ecole Polytechnique Fédérale de Lausanne
2013	**Graduated with honours** Slovak University of Technology, Slovakia
2011	**Dean's award** awarded with Bachelor Degree for excellent results Slovak University of Technology, Slovakia
2010	**Best student of 2010** awarded at Faculty of Electrical Engineering and Information Technology Slovak University of Technology, Slovakia

Languages

- **Slovak** (mother tongue)
- **English** (fluent)
- **German** (good)
- **French** (basic)

Skills and Knowledge

- **Extensive knowledge of modern cryptography and security** gained through undergraduate and graduate studies with the major in cryptography and security, the study exchange at the Ruhr Universität Bochum (focused on cryptography and security) and currently the study in the Security and Cryptography Laboratory at the Ecole Polytechnique Fédérale de Lausanne.
- **Experience with programming in C and C++** obtained mainly through various projects in bachelor and master studies. Co-author of the reference and optimized implementations of the CAESAR candidate OMD.
- **Practical Experience with administration of operating systems and IT security** obtained through practical sessions during master studies and practical experience form the ING Insurance Company.
- **Extensive knowledge of authenticated encryption and provable security** which is the main research topic of the author.
- **Experience with NFC programming on Android and Linux.** Obtained through the preparation of a working demo of relay attacks on NFC-credit card payments for the open house of School of computer and communication science 2014 at Ecole Polytechnique Fédérale de Lausanne.