Under Pressure: Security of Caesar Candidates beyond their Guarantees

Serge Vaudenay and Damian Vizár

EPFL, Switzerland

Abstract. The Competition for Authenticated Encryption: Security, Applicability and Robustness (CAESAR) has as its official goal to "identify a portfolio of authenticated ciphers that offer advantages over AES-GCM and are suitable for widespread adoption." Each of the 15 candidate schemes competing in the currently ongoing 3rd round of CAESAR must clearly declare its security claims, i.a. whether or not it can tolerate nonce misuse, and what is the maximal data complexity for which security is guaranteed. These claims appear to be valid for all 15 candidates. Interpreting "Robustness" in CAESAR as the ability to mitigate damage even if security guarantees are void, we describe attacks with birthday complexity or beyond, and/or with nonce reuse for each of the 15 candidates. We then sort the candidates into classes depending on how powerful does an attacker need to be to mount (semi-)universal forgeries, decryption attacks, or key recoveries. Rather than invalidating the security claims of any of the candidates, our results provide an additional criterion for evaluating the security that candidates deliver, which can be useful for e.g. breaking ties in the final CAESAR discussions.

Keywords: Authenticated Encryption · CAESAR Competition · Forgery · Decryption Attack · Birthday Bound · Nonce Misuse

1 Introduction

Authenticated encryption (AE) is a symmetric key primitive that simultaneously ensures confidentiality, integrity and authenticity of encrypted messages [BR00, KY01]. Most of existing AE schemes also allow to authenticate a public string, the associated data, along with the message and are called schemes for AE with associated data (AEAD)[Rog02]. During the decade and a half of its existence, AE has been not just a frequent research object but also a frequently used tool (e.g. in IEEE 802.11i, IPsec ESP and IKEv2, NIST SP 800-38D, ANSI C12.22, and ISO/IEC 19772:2009), especially because most practical applications of symmetric key cryptography require both confidentiality and integrity at the same time.

In 2013, the Competition for Authenticated Encryption: Security, Applicability and Robustness (CAESAR) was announced. The reason for launching the competition was, in part, a startling amount of recently discovered issues with applications of symmetric cryptography, as well as with the most popular AE schemes CCM [WFH03, Jon03] and GCM [MV04]. The security misses in the applications of symmetric encryption constituted practically exploitable vulnerabilities [Berb] while in the case of CCM and GCM, concerns were expressed about their applicability [RW03], their security proofs [IOM12] or their susceptibility to devastating attacks when the usage conditions were violated [Fer05, Jou06].

Thus CAESAR's main goal has been set to "identify a portfolio of authenticated ciphers that offer advantages over AES-GCM and are suitable for widespread adoption" [Bera]. GCM instantiated with the AES blockcipher has been used as 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. Out of 57 submissions to the first round of CAESAR, 15 candidates still compete in the 3rd round [CAE]. All of the 3rd

Table 1: Classification of 3rd round CAESAR candidates based on their claimed security guarantees w.r.t the nonce misuse and quantitative security. Here "birthday bound" informally refers to about 2^{64} processed bits. For security in presence of nonce misuse, we consider MRAE [RS06], OAE [FFL12] or RAE [HKR15]. For each candidate, we consider an instance with 128-bit secret key.

	up to "birthday bound"	**beyond "birthday bound"**
unique nonces	OCB, NORX, Jambu, CLOC&SILC	Tiaoxin, Morus, Keyak, Ketje, Deoxys I&II, Ascon, AEGIS, ACORN
nonce misuse	Deoxys II, COLM, AEZ	-

round candidates' security claims are supported by a solid amount of cryptanalysis and/or security proofs, and are generally believed to be sound.

Birthday bound and nonce-misuse. All of CAESAR candidates must accept so called public message number, a.k.a. nonce along with the secret key, AD and the message as an input. The nonce is akin to an initialization vector, and it can be assumed to have a unique value for every encryption query. The candidates are allowed to request that the nonce must not repeat in order for their security guarantees to apply. This is the case for 12 3rd round CAESAR candidates. AEZ and Deoxys guarantee no degradation of authenticity, and the minimal (and unavoidable) degradation of confidentiality[1] even if the nonces are misused, i.e. repeated. COLM guarantees a weaker version of confidentiality protection in presence of nonce misuse. Each candidate must also specify how much data can be securely processed with a single secret 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 paper, we abuse the term "birthday bound" and let it refer to this data complexity - about $\gamma \cdot 2^{64}$ processed bits for a small γ.

We sort the 3rd round candidates, as well as CCM and GCM, into four classes based on their security claims w.r.t. the nonce misuse and quantitative security in Table 1. We consider a scheme to claim security against nonce reuse if it targets MRAE [RS06], OAE [FFL12] or RAE [HKR15] security. For each candidate, we consider an instance with 128-bit secret key.

Robustness: (in)security beyond guarantees. All CAESAR candidates clearly state what security properties do they guarantee as long as the usage conditions, such as respecting the nonces or data limits, are respected. However, they give little or no information on what exactly happens when their guarantees become void.

This is what we aim to determine in this work. We take the liberty to interpret robustness of AE schemes as the ability to resist powerful attacks, possibly beyond the limitations guaranteed by the designers, and analyze the security of all 15 3rd round CAESAR candidates against attacks at the birthday bound or beyond, and against nonce-misuse attacks. We consider secret keys of 128 bits, and we informally call "birthday bound" a complexity with order of magnitude 2^{64}, in order for the results for different candidates to be comparable.

Our contribution. For each candidate we present one or more attacks. We sort the CAESAR candidates into five 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 birthday bound. **(B)** Those others for which we have a nonce-respecting universal forgery *and* a decryption attack beyond the birthday bound, but below exhaustive search. **(C)** Those others for which we

[1] As the encryption is required to be a deterministic algorithm, repeating all inputs unavoidably means repeating the ciphertexts as well.

Table 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) and (E) are listed from top to bottom. Reusability means that forging/decrypting multiple ciphertexts without significantly increasing the time/data complexity, and the comments "(but N, A)", "(but N)" and "(but A)" mean that the reusability is limited to fixed values of the listed parameters. The values in the column "nonce-misuse" indicate maximal number of times any nonce is used (so $1 \Rightarrow$ nonce respecting) and q denotes the number of forgeries intend to do in a single attack.

algorithm	**type of attack**	**nonce-reuse**	**# queries**	**reusable**
AES-GCM ($\|N\| > 128$ **only)**	universal forgery	1	$3 \cdot 2^{64}$	yes
AEZ	key recovery	1	$3 \cdot 2^{64}$	
OCB	universal forgery & CCA decryption	1	2 (one of length 2^{64})	yes
AES-OTR	universal forgery & CPA decryption	1	2 (one of length 2^{64})	yes
CLOC	universal forgery & CPA decryption	1	2^{80}	yes
AES-GCM	universal forgery & CPA decryption	2	2	yes
CCM	CPA decryption	2	1	
DEOXYS-I	universal forgery & CCA decryption	3	3	yes
Tiaoxin	key recovery	30	30	
AEGIS-128	universal forgery & CPA decryption	15	15	yes (but N, A)
ACORN-128	universal forgery & CPA decryption	300	300	yes (but N, A)
CLOC & SILC	CPA decryption	2	1	no
Ketje Sr	key recovery	50	50	
MORUS 640	universal forgery & CPA decryption	8	8	yes (but N)
NORX32-4-1	CPA decryption	$\|C\|/384$	$\|C\|/384$	no
Ascon-128	CPA decryption	$\|C\|/64$	$\|C\|/64$	no
Lake Keyak	CPA decryption	$\|C\|/1344$	$\|C\|/1344$	no
COLM	semi-universal forgery	$1+q$	2^{64}	yes (but N, A)
JAMBU	universal forgery	1	2^{64} (decryption)	no
Deoxys-II	semi-universal forgery & CCA decryption	2^m	2^{128-m}	yes (but A)

have a forgery *or* a decryption attack with small complexity, possibly with nonce-misuse. **(D)** Those others for which we have a forgery *or* a decryption attack at the birthday bound, possibly with nonce-misuse. **(E)** Remaining ones.

Our results are summarized in Table 2, where the categories (A), (B), (C), (D) and (E) are listed in this order. For each candidate, we indicate what type of attack is mounted, what is the query complexity[2], whether it needs nonce misuse, and whether it is reusable. We call a forgery or decryption attack reusable, if forging/decrypting multiple ciphertexts does not significantly increase the time/data complexity of the attack. The comments "(but N, A)", "(but N)" and "(but 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 2 succeed with high probability.

The contribution of this result is twofold. Firstly, Table 1 shows a situation that is quite different from the one suggested by Table 2. Based on the former, one could think that any of the schemes within the same category would provide the same kind of security. Table 2 does however suggest, that this may not at all be the case, and that the exact security provided by candidates with the same kind of guarantees differs. This can be very useful to break ties at the end of 3rd round of CAESAR competition.

Secondly, some of these attacks can be viewed as disturbingly powerful (e.g. key recoveries). Taking into consideration the circumstances that led to the start of CAESAR competition, it is debatable whether schemes that succumb to such attacks are robust enough (in the specific sense we picked for this paper) to be recommended as CAESAR finalists.

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

[2]The time and memory complexities of the attacks mentioned in the table are small multiples of the query complexity.

what degree will the security of respective candidates deteriorate *after* the guarantees become void.

Related work. The (in)security of GCM mode was treated by a number of works [IOM12, Saa12, HP08, PC15], in particular Joux authored the "forbidden" nonce misusing attack [Jou06]. Collision attack similar to ours, or inspiring ours, were described for previous versions of AEZ by Fuhr et al. [FLS15], and Chaigneau and Gilbert [CG16]. Collision attack 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 shceme of an earlier version of OTR [BS16], Huang and Wu described a collision based forgery [HW]. Mileva et al. describe a nonce misusing distinguisher attack for MORUS [MDV16]. The collision-based forgeries on NORX, Ascon and Keyak are matching Lemma 2 of the work on provable generic security of full-state keyed duplex by Daemen et al. [DMA17].

Acknowledgements. We would like to thank all CAESAR designers who provided us with their feedback. We would like to thank the Ascon team for pointing out that generic attacks with the same time but much lower data complexity than our forgery exist, and the Deoxys team for suggesting a better way to measure adversarial resources for nonce misuse.

Organization of the paper. In Section 2 we introduce notations that are used in the paper. Then we address CCM, GCM abd each CAESAR candidate in a separate section.

2 Preliminaries

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 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 tag length (or the ciphertext expansion/stretch, in general).

Syntax and attack model. Each of the candidates defines an encryption algorithm $\mathcal{E}$ that maps a tuple (K, N, A, M) to a ciphertext. For most candidates, the ciphertext consists of a core ciphertext C and a tag T. The candidates also define a decryption algorithm $\mathcal{D}$ that maps (K, N, A, C) or (K, N, A, C, T) to a message M or to an error symbol $\perp$, if the authentication fails.

For our attacks, we assume the adversary can make black-box oracle queries to the encryption algorithm, and possibly to the decryption algorithm as well, both initialized with an unknown key. The adversary is able to choose all inputs freely, but it may be forced to use unique nonces for encryption queries.

In a decryption attack, we are given a tuple (N, A, C, T) produced with an unknown message and we try to recover the underlying message without making the decryption query (N, A, C, T). In a forgery attack, we aim at producing a valid tuple (N, A, C, T) that authenticates correctly, but was not obtained from a previous encryption query.

Notations. 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. We let $M_1, \ldots, M_\ell \xleftarrow{n} M$ denote the partitioning of a string M into blocks of n bits, except for $1 \leq |M_\ell| \leq n$, such that $\ell = \lceil |M|/n \rceil$. We let $|M|_n = \lceil |M|/n \rceil$. We let ε denote the empty string and $|X|$ the length of a string X in bits. For two strings X, Y with $|X| = |Y|$, we let $X \& Y$ denote the bitwise AND of X and Y. 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 a block

size, when a block size is implicit from the context. We let $\mathsf{msb}_a(X)$ denote the a most significant bits of a string X, and similar applies to lsb_a. We let $\mathrm{enc}_n(a)$ denote the n-bit canonical encoding of an integer $0 \leq a \leq 255$.

3 AES-CCM

CCM [WFH03] 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 $U = \mathrm{CBC}_K(B)$ with (with no IV for CBC) 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 $T = \mathsf{msb}_\tau(U \oplus E_K(\mathsf{I}(N, 0)))$.

CPA decryption (nonce-misuse, tiny complexity). If we are able to force a nonce to be reused with the same secret key, then decrypting a ciphertext-nonce pair N, C is simply achieved by querying $(N, \varepsilon, 0^{|C|})$ to get the ciphertext C' and the tag, and computing the message $M \leftarrow C \oplus C'$.

Semi-universal forgery (nonce-misuse, birthday bound). We assume that $\tau = 128$. This forgery is semi-universal, we can only choose M and have no control over N and A. We make 2^{64} queries (N^i, A^i, M') where N^i-s are distinct, each A^i is a distinct randomly chosen string of 240 bits (this makes sure that length of N and A encoded in B is exactly 3 blocks), and $M' \neq M$ but $|M'| = |M|$. We expect to find $i \neq j$ with $T^i = T^j$, for which we deduce that the internal states of the CBC MAC just after processing of AD collided. Then we make a query (N, A^i, M), get C, T, and forge with (N, A^j, C, T).

4 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

$$\mathrm{GHASH}_L(A, C) = \bigoplus_{i=1}^{\ell} L^{\ell-i+1} \cdot X_i, \qquad \text{with} \qquad X = A\|0^*\|C\|0^*\|\mathrm{enc}_{64}(|A|)\|\mathrm{enc}_{64}(|C|)$$

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 $I \leftarrow \mathrm{GHASH}_L(\varepsilon, N)$ if $|N| \neq 96$. Then we compute the ciphertext C as counter mode encryption of M, using[3] $\mathsf{inc}(Y) \mapsto \mathsf{msb}_{96}(Y)\|(\mathsf{lsb}_{32}(Y) + 1)$ as the incrementation function and $\mathsf{inc}(I)$ as the initial counter value. Then we compute the tag as $T = \mathsf{msb}_\tau(\mathrm{GHASH}_L(A, C) \oplus E_K(I))$.

CPA decryption (nonce-misuse, tiny complexity). The nonce-misuse decryption attack on CCM is applies to GCM as well.

Universal forgery (nonce misuse, tiny complexity). This attack has been first described by Joux as the "forbidden attack" [Jou06] (as nonce misuse is "forbidden" by GCM's security model). The core of the attack is recovering the derived key L, which makes forging very easy. We assume that $\tau = 128$.

We make two queries (N, ε, M^i) with all M^1, M^2 random distinct strings of 128 bits, and get C^i, T^i for $i = 1, 2$. We then compute the polynomial over $\mathsf{GF}(2^{128})$ $P(\Lambda) = (C_1^1 \oplus C_1^2) \cdot \Lambda^2 \oplus (T^1 \oplus T^2)$. As T_1 and T_2 are computed with the same nonce, $T^1 \oplus T^2$ will be an XOR of the underlying outputs

[3] We abuse the notation slightly; the incrementation is done with integer representation of $\mathsf{lsb}(Y)$.

of GHASH and thus L will be a root of $P(\Lambda)$. Moreover, L will be the only root as squaring yields a bijection over $\mathsf{GF}(2^{128})$. Then, to forge a ciphertext for a tuple (N, A, M), we make a query (N, A', M') with arbitrary A' and M' such that $|M| = M'$. We forge with (N, A, C, T) where $C = C' \oplus M' \oplus M$ and $T = T' \oplus \mathrm{GHASH}_L(A', C') \oplus \mathrm{GHASH}_L(A, C)$.

Universal forgery (nonce respecting, birthday bound, $|N| > 128$). If nonces longer than 128 bits are allowed, it is possible to recover 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]. We assume that $\tau = 128$. We make 2^{64} encryption queries (N^i, ε, M^i) with all N^i random distinct strings of 256 bits, $M^i = B \| M_2^i$ for a fixed block B and each M_2^i a random distinct string of 128 bits and get C^i, T^i-s. We expect to find $i \neq i'$ such that $C_1^i = C_1^{i'}$, which implies $\mathrm{GHASH}_L(\varepsilon, N^i) = \mathrm{GHASH}_L(\varepsilon, N^{i'})$. We construct the polynomial $P(\Lambda) = (C_2^i \oplus C_2^{i'}) \cdot \Lambda^2 \oplus (T^i \oplus T^{i'})$ for this pair, and recover L as in the previous attack.

5 AEZ v5

AEZ encryption [HKR] (in the AEZ-core case: the general one) has the following structure

$$\mathsf{Encrypt}(K, N, A, \tau, M) = f(I, J, L, \Delta, M)$$

where K is the secret key, N is a nonce, A is the associated data, τ is the stretch, M is the plaintext, $(I, J, L) = \mathsf{KDF}(K)$, 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 total break attacks. All binary inputs can have an arbitrary length and are internally processed in blocks of 128 bits.

Furthermore, whenever we encrypt the same message M with the same key K, any collision on the hash function results in getting the same ciphertext. Conversely, if the message is long enough, any collision on the ciphertext when encrypting the same plaintext with the same key must imply a collision on the hash function. So, it is easy to detect collisions on the hidden variable Δ in a chosen message attack.

In what follows, we denote $k = (0, J, I, L, 0)$ 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 and integer times a block (badly) denotes the classical $\mathsf{GF}(2^{128})$ multiplication.

***I*, *J*, *L*-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 [HKR] 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: pick an arbitrary block c and a long enough (a few blocks) message M
2: make chosen message encryption queries with input (N, A, τ, M), for a random (but fresh) N and $A = N \oplus c$, until we see a ciphertext collision with N and N'
3: deduce $J = N \oplus N' \oplus c$

The Chaigneau-Gilbert attack requires little efforts to be adapted to AEZ v5 but it can recover I and L with a nonce-misuse attacks. Here, we notice that we can use a feature of AEZ allowing to

use nonces of several blocks to have a similar attack as the one above [HKR]. We now use an empty A and N of several blocks. If $|N|_{128} = 2$, then following the AEZ v5 specifications 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 limit ourselves to encryption queries where $N_2 = N_1 \oplus c$ for a fixed block c and we see that a ciphertext collision with N and N' such that yields $L = N_1 \oplus N_1' \oplus c$.Thus we recover L in a similar attack as before.

Next, we see that when $|N|_{128} = 9$, the hash function H becomes

$$\begin{aligned} h_k(\tau,(N_1,\ldots,N_9)) &= E_K^{3,1}(\tau) \oplus E_K^{4,1}(N_1) \oplus \cdots \oplus E_K^{4,9}(N_9) \\ &= E_K^{3,1}(\tau) \oplus \mathsf{AES4}_k(N_1 \oplus 4J \oplus 2I \oplus L) \oplus \cdots \oplus \\ &\quad \mathsf{AES4}_k(N_7 \oplus 4J \oplus 2I \oplus 7L) \oplus \mathsf{AES4}_k(N_8 \oplus 4J \oplus 2I) \oplus \\ &\quad \mathsf{AES4}_k(N_9 \oplus 4J \oplus 4I \oplus L) \end{aligned}$$

We limit ourselves to encryption queries where $N_2, \ldots, N_8$ are constant and $N_9 = N_1 \oplus c$ for a constant block c and we see that a ciphertext collision with N and N' yields $6I = N_1 \oplus N_1' \oplus c$. We recover I in a similar attack as we mounted for J. So, we recover I, J, L with a nonce-respecting chosen message attack at the birthday bound.

6 OCB3 (OCB v1.1)

OCB [KR] (a.k.a. OCB3) uses AES-128 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

$$C_i = E_K(M_i \oplus \Delta_i) \oplus \Delta_i$$

with

$$\Delta_i = R \oplus \gamma_i \cdot L$$

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 $1 \le i \le 2^{120}$. 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 block M_ℓ of M is complete and $\tau = 128$, the tag is computed as

$$T = E_K\left(2^{-1} \cdot L \oplus \Delta_\ell \oplus \bigoplus_{i=1}^{\ell} M_i\right) \oplus H_K(A)$$

where (assuming the last block of A is complete)

$$H_K(A) = \bigoplus_i E_K(A_i \oplus \gamma_i \cdot L)$$

and $\cdot$ denoting the multiplication in $\mathsf{GF}(2^{128})$. The attacks we describe can be easily generalized to the case when the last block of M and the last block of A are not 128 bits long

***L*-recovery attack.** An attack by Ferguson [Fer02] allows to recover 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

- define a permutation $\pi : \{1, \ldots, \ell\} \to \{1, \ldots, \ell\}$ as $\pi(i) = (i + 1 \bmod \ell) + 1$,
- construct a message M with $M_i = M'_{\pi(i)} \oplus \gamma_i \cdot L \oplus \gamma_{\pi(i)}$ for every i

Then, we make a query with (N, A, M) to get (C, T). We compute C' by $C'_i = C_{\pi^{-1}(i)} \oplus (\gamma_i \oplus \gamma_{\pi^{-1}(i)}) \cdot L$ for every i and it gives a valid encryption (C', T) of (N, A, M'). 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: first, encrypt a random message M with $A = \varepsilon$ such that $|M|_{128} = \ell$ and $\bigoplus_{i>1} M_i = (2^{-1} \oplus \gamma_1 \oplus \gamma_\ell) \cdot L$. This way, we have $T = E_K(M_1 \oplus R \oplus \gamma_1 \cdot L) = C_1 \oplus R \oplus \gamma_1 \cdot L$ from which we deduce R. Additionally, we obtain ℓ pairs $(x_i, E_K(x_i))$ from the (M_i, C_i) pairs, with known $x_i = M_i \oplus R \oplus \gamma_i \cdot L$. When $\ell = 2^{14}$ (so a message of 256KB), we expect to have one index i for which x_i is of form $x_i = 0^7 1 \| N'$ for some N' ending with six zero bits. (We can even make sure this happens by taking $M_i = (a_i \| B \| b_i) \oplus \gamma_i \cdot L$ with a_i of eight bits, b_i of six bits, and $i \mapsto (a_i \| b_i)$ being surjective onto the 14-bit strings.) So, we can deduce $R' = E_K(0^* 1 \| N')$ for the unused nonce N'. Using this (N', R') pair, we can make a nonce-respecting query with nonce N' to obtain many new $(x'_i, E_K(x'_i))$ pairs with all x'_i chosen. We can even prepare the next nonces to be respectfully used and we implement an E_K oracle this way. This attack is nonce-respecting with tiny complexity but using L. It uses an encryption query with a message of 256KB for the first time it is run. Then, it uses a single encryption query of size corresponding to the values we want to encrypt (plus one block).

CCA decryption attack for messages of odd length (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 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. So, a CCA decryption query with (N', A, C', T) gives M' from which we deduce M.

7 AES-OTR v3.1

AES-OTR v3.1 with parallel AD processing [Min] produces a tag T which is $\mathsf{msb}_\tau(\mathsf{TA} \oplus \mathsf{TE})$ where TA and TE are tags for the associated data A and the pair (N, M), respectively. We assume that $|N| = 120$. Interestingly, 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-128 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

$$\mathsf{TE} = E_K\left(7 \cdot 2^{\ell-1} \cdot 3 \cdot L \oplus \bigoplus_{i=1}^{\ell} M_{2i}\right)$$

***L*-recovery attack (nonce-misuse).** If we use the same nonce N (so with nonce misuse), L is fixed. We can encrypt messages of four blocks and look for a matching

$$C_1 \oplus M_2 = C'_3 \oplus M'_4$$

It occurs at the birthday bound. Clearly, this is equivalent to

$$M_1 \oplus L = M'_3 \oplus 2 \cdot L$$

which reveals L.

***L*-recovery attack (nonce respecting).** We encrypt a huge random message (with $|M|_{128} \approx 2^{64}$) with a nonce N and look for an internal collision

$$C_{2i} \oplus M_{2i-1} = C_{2j} \oplus M_{2j-1}$$

with $i \neq j$ Clearly, this is equivalent to

$$C_{2i-1} \oplus 2^{i-1} \cdot 2 \cdot L = C_{2j-1} \oplus 2^{j-1} \cdot 2 \cdot L$$

which reveals L for this N. We also 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: set $M_{2i-1} = x_i \oplus 2^{2i-1} \cdot L$ and M_{2i} arbitrarily for $i = 1, \ldots, r$. Then, $E_K(x_i) = M_{2i} \oplus C_{2i-1}$.

The first time we use this oracle may be a nonce misuse case or not (depending on the way we obtain N, L), but then we can always add the block $x_{r+1} = \epsilon(\tau)\|1\|N'$ to the list to prepare an (N', L') pair for the next execution of the E_K oracle, with N' a fresh nonce.

The oracle E_K can be used to mount universal forgeries and CPA decryption attacks with tiny complexity (but needs L).

***Q*-recovery attack (nonce-misuse).** Assuming all encryption queries we make use the same N and M (so with nonce misuse), a collision on the tag is equivalent to a collision on TA. We make queries with associated data of three blocks with A_3 and $A_1 \oplus A_2 = \delta$ constant. Clearly, a collision between A and A' is equivalent to

$$E_K(A_1 \oplus Q) \oplus E_K(A_1 \oplus \delta \oplus 2 \cdot Q) = E_K(A'_1 \oplus Q) \oplus E_K(A'_1 \oplus \delta \oplus 2 \cdot Q)$$

which can be caused by

$$A'_1 = A_1 \oplus \delta \oplus 3 \cdot Q$$

Once this event occurs (at the birthday bound), we can deduce Q.

8 CLOC and SILC

CLOC and SILC v3 [IMG+] use the nonce N of 96 bits. 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 and the blockcipher E is AES-128.

CPA decryption attack (tiny complexity, nonce-misuse). To decrypt (N, A, C, T), we can first set an arbitrary message M' of one block and make a query (N, A, M'). We obtain C' and deduce $M_1 = C'_1 \oplus C_1 \oplus M'_1$. Then, we can set an arbitrary message M' of two blocks with $M'_1 = M_1$ and make a query (N, A, M'). We obtain C' and deduce $M_2 = C'_2 \oplus C_2 \oplus M'_2$. We proceed iteratively until all blocks of M are recovered. This is a nonce-misuse CPA attack with very low complexity.

E_K oracle with input of msb equal to 1 (tiny complexity, nonce-misuse). When x has its most significant bit set to 1, we can easily compute $E_K(x)$ with two nonce-misuse queries: make one first query with an arbitrary (N, A, M) with M of at least one block, then get (C, T), then make a nonce-misuse query with (N, A, M') with M' arbitrary such that $M'_1 = M_1 \oplus C_1 \oplus x$. We get (C', T') and $C'_2 \oplus M_2 = E_K(x)$.

As both CLOC and SILC start with the first E_K evaluation with an input having an msb set to zero, this cannot be used to make a forgery.

Existential forgery. If we make many (about 2^{64}) encryption queries (N^i, A, M^i) with random N^i and M^i and fixed A in a nonce-respecting way, we find $i \neq j$ with $C^i = C^j$, which implies a collision on V. Then, (N^i, A, C^j, T^j) is a valid forgery for the query (N^i, A, M^j).

E_K oracle in CLOC (nonce-respecting, beyond birthday bound). We can make many encryption queries (N^i, A, M^i) with a fixed A until we have a collision $M^i_1 \oplus C^i_1 = M^j_2 \oplus C^j_2$ with $i \neq j$. This indicates that $V = \mathsf{fix1}(C'_1)$ so we deduce the V value for a random nonce N with A. In CLOC, we have

$$V = f_1(W \oplus \mathsf{ozp}(\mathsf{param} \| N))$$

for some W depending on K and A only, and some easy-to-invert function f_1. So, from the knowledge of V, we can deduce W from which we can compute the $\bar{V}$ value associated to an arbitrary $\bar{N}$ and the same A. Hence, we can make chosen queries $(\bar{N}, A, \bar{M})$ that will use a predictable value $\bar{V}$ in a nonce-respecting way.

If we want to compute $E_K(x)$, if $f_1^{-1}(x) \oplus W$ is of form $\mathsf{ozp}(\mathsf{param} \| \bar{N})$, we can just make a random query with this $\bar{N}$ then deduce $E_K(x) = M_1 \oplus C_1$. If we do not have 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 could run the previous attack 2^{16} times to get more chances. So, with complexity 2^{80}, we collect many W_i to make sure that at least one is such that $f_1^{-1}(x) \oplus W_i$ is of the correct format.

This attack does not work on SILC, in which W depends on the nonce as well.

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 Deoxys v1.41

Deoxys v1.41 [JNP] has two instantiations: Deoxys-I, which claims no nonce-misuse security, and Deoxys-II, which claims to resist nonce misuse attacks. The internal design is very similar to OCB, except that it relies on an ad-hoc tweakable encryption E^T_K instead of AES with the input and output masks Δ. The block size of E is 128 bits. 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

$$T = E_K^{1\|N\|\ell}\left(\bigoplus_{i=1}^{\ell} M_i\right) \oplus H_K(A)$$

(when the last block M_ℓ of M is complete). In Deoxys-II, we have

$$T = E_K^{1\|0^4\|N}\left(H_K(A) \oplus \bigoplus_{i=1}^{\ell} E_K^{0\|(i-1)}(M_i)\right)$$

(when the last block M_ℓ of M is complete) and

$$C_i = M_i \oplus E_K^{1\|(T\oplus(i-1))}(0^8\|N)$$

Universal forgery and CCA decryption attack of Deoxys-I (tiny complexity, nonce-misuse). Deoxys-I is susceptible to low-complexity nonce-misuse attacks. Indeed, given $\mathcal{E}_K(N, A, M) = (C, T)$ and $\mathcal{E}_K(N, A', M) = (C, T')$, we deduce $T \oplus T' = H_K(A) \oplus H_K(A')$. Then, $\mathcal{E}_K(\bar{N}, A, \bar{M}) = (\bar{C}, \bar{T})$ can be transformed into $(\bar{C}, \bar{T} \oplus T \oplus T')$ which is a valid forgery of $\mathcal{E}_K(\bar{N}, A', \bar{M})$. Furthermore, we can decrypt $(\bar{N}, A, \bar{C}, \bar{T})$ in a CCA attack by querying $(\bar{N}, A', \bar{C}, \bar{T} \oplus T \oplus T')$ which has the same decryption.

Semi-universal forgery and CCA decryption attack of Deoxys-II (reusable, nonce-misuse). We assume that every nonce can be repeated 2^m times. We make 2^{128-2m} batches of 2^m queries. In the i^{th} batch, we query $\mathcal{E}_K(N^i, A^{i,j}, M) = (C^{i,j}, T^{i,j})$ for $j = 1 \ldots, 2^m$ with N^i fixed in the batch, M fixed globally, and random $A^{i,j}$. We then find i and $j \neq j'$ such that $A^{i,j} \neq A^{i,j'}$ and $T^{i,j} = T^{i,j'}$. We deduce that $H_K(A^{i,j}) = H_K(A^{i,j'})$. Then, assume we want to make a forgery for $(\bar{N}, A^{i,j}, \bar{M})$ (i.e. $A^{i,j'}$ is imposed). We can query $\mathcal{E}_K(\bar{N}, A^{i,j'}, \bar{M}) = (\bar{C}, \bar{T})$ and deduce the forgery $\mathcal{E}_K(\bar{N}, A^{i,j}, \bar{M}) = (\bar{C}, \bar{T})$. Similarly, we can decrypt $(\bar{N}, A^{i,j}, \bar{C}, \bar{T})$ by making a CCA decryption query on $(\bar{N}, A^{i,j'}, \bar{C}, \bar{T})$. 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.

10 Tiaoxin-346

Tiaoxin-346v2.1 [Nik] loads the key K and the nonce IV in a state, then applies a reversible transformation. Each block is represented by a 4×4 matrix of bytes. The state consists of three arrays $T[3]$, $T[4]$, and $T[6]$, of respectively 3, 4, and 6 blocks. Given a state $(T[3], T[4], T[6])$, we load the plaintext by pairs (M_0, M_1) of blocks and output two blocks (C_0, C_1). This operation works in five steps:

1. $T[3] \leftarrow R(T[3], M_0)$
2. $T[4] \leftarrow R(T[4], M_1)$
3. $T[6] \leftarrow R(T[6], M_0 \oplus M_1)$
4. $C_0 = T[3]_0 \oplus T[3]_2 \oplus T[4]_1 \oplus (T[6]_3 \& T[4]_3)$
5. $C_1 = T[6]_0 \oplus T[4]_2 \oplus T[3]_1 \oplus (T[6]_5 \& T[3]_2)$

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:

$$A(x) = \mathsf{MixColumns}(\mathsf{ShiftRows}(\mathsf{SubBytes}(x)))$$

Nonce-misuse key recovery. Assume that we encrypt two messages $M = (M_0, M_1, M_2, M_3)$ and $\bar{M} = (\bar{M}_0, \bar{M}_1, \bar{M}_2, \bar{M}_3)$ starting from the same state (with nonce-misuse), with $\bar{M}_i = M_i \oplus \Delta$ for $i = 0, 1, 2, 3$. The first encryption defines

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. $C_3 = T''[6]_0 \oplus T''[4]_2 \oplus T''[3]_1 \oplus (T''[6]_5 \& T''[3]_2)$

The second encryption defines

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. $\bar{C}_1 = T'[6]_0 \oplus \bar{T}'[4]_2 \oplus \bar{T}'[3]_1 \oplus (T'[6]_5 \& \bar{T}'[3]_2)$
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

$$\begin{aligned}
\bar{T}'[3] \oplus T'[3] &= (\Delta, 0, 0) \\
\bar{T}''[3] \oplus T''[3] &= (0, A(T'[3]_0) \oplus A(T'[3]_0 \oplus \Delta), 0) \\
\bar{T}'[4] \oplus T'[4] &= (\Delta, 0, 0, 0) \\
\bar{T}''[4] \oplus T''[4] &= (0, A(T'[4]_0) \oplus A(T'[4]_0 \oplus \Delta), 0, 0)
\end{aligned}$$

Hence

$$\begin{aligned}
\bar{C}_2 \oplus C_2 &= A(T'[4]_0) \oplus A(T'[4]_0 \oplus \Delta) \\
\bar{C}_3 \oplus C_3 &= A(T'[3]_0) \oplus A(T'[3]_0 \oplus \Delta)
\end{aligned}$$

These two equations can be transformed to the form $\gamma = \mathsf{SubBytes}(X) \oplus A(X \oplus \Delta)$ where $\gamma = \mathsf{ShiftRows}^{-1}(\mathsf{MixColumns}^{-1}(\bar{C}_i \oplus C_i))$. This will suggest either 2 or 4 possible values for each byte of $T'[4]_0$ (or $T'[3]_0$). By making another query with $\tilde{M}$ (with the same nonce) with $\tilde{M}_i = M_i \oplus \tilde{\Delta}$ for $i = 0, 1, 2, 3$ and $\tilde{\Delta}$, we build similar equations, and each byte of $T'[4]_0$ (or $T'[3]_0$) will be uniquely determined as an intersection of values suggested by the two equations.

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 can invert the initialization and obtain K.

11 AEGIS v1.1

AEGIS v1.1 [WP] 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 one tag is extracted from the result.

Universal forgery and decryption attack (tiny complexity, nonce-misuse). Assume we want to forge the encryption of (N, A, M) or that we want to decrypt (N, A, C, T). We can run an attack to recover the state after processing A with nonce N. With this state, we can run the encryption or decryption ourselves. To recover the state, we will make nonce-misuse encryption queries with the same (N, A) and changing messages M^i. We notice that the first block M_1^i of M^i only directly influences S_0, so for $i \neq j$ we always have $C_2^i \oplus M_2^i = C_2^j \oplus M_2^j$. But a difference becomes visible in C_3. Indeed, we can see that the difference $(C_3^i \oplus M_3^i) \oplus (C_3^j \oplus M_3^j)$ (associated to $M_1^i \neq M_1^j$) is equal to the difference $R(R(S_4) \oplus S_0 \oplus M_1^i) \oplus R(R(S_4) \oplus S_0 \oplus M_1^j)$ (where $R(S_4) \oplus S_0 = S_0'$).

Since R is a single AES round, this is equivalent with a differential equation on the SubBytes layer of AES. Similarly as for Tiaoxin, we can make 3 queries that yield two distinct differences $\Delta_1^{i,j} = M_1^i \oplus M_1^j$ and uniquely determine S_0'. Using the same strategy in varying M_2, we deduce $R(S_4')$ and so S_4' with another 3 queries. We can further iterate this strategy with M_3, M_4 and M_5 and we fully reconstruct the state with 15 queries in total.

The possibility of such an attack is mentioned in the AEGIS v1.1 specifications [WP].

12 ACORN v3

ACORN v3 [Wu] 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

$$\begin{aligned} o &= |A| + 256 \\ S_{i+1+o} &= \mathsf{StateUpdate128}(S_{i+o}, M_i, 1, 0) \\ C_i &= M_i \oplus \mathsf{ks}_{i+o} \\ \mathsf{ks}_{i+o} &= S_{i+o,12} \oplus S_{i+o,154} \oplus \mathsf{maj}(S_{i+o,235}, S_{i+o,61}, S_{i+o,193}) \oplus \mathsf{ch}(S_{i+o,230}, S_{i+o,111}, S_{i+o,66}) \end{aligned}$$

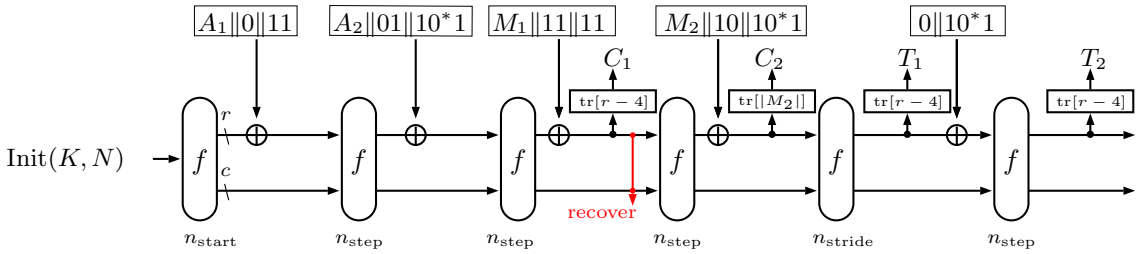

Figure 1: The encryption algorithm of Ketje [BDP+a] with two-block A and two-block M as input. We let $tr[x]$ denote truncation to x leftmost bits.

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 $k + j \leq 292$.

Universal forgery and decryption attack (tiny complexity, nonce-misuse). Assume we want to forge the encryption of (N, A, M) or that we want to decrypt (N, A, C, T). We can run an attack to recover the internal state S_o just after processing A with the nonce N. With this state, we can run the encryption or decryption ourselves.

To recover the state, we will make nonce-misuse encryption queries with the same (N, A) and changing messages. By changing M_0^j, we can see that ks_{i+o}^j is constant for $i = 0, \ldots, 57$. We can easily see that for $j \neq j'$, $\mathsf{ks}_{58+o}^j \oplus \mathsf{ks}_{58+o}^{j'} = S_{58+o,61} \oplus S_{58+o,193}$, which is a linear function in S_o. So, we have one linear equation in the bits of S_o. We can do the same by fixing M_0^j and changing M_1^j and so on to recover more equations. We could reconstruct S_o like this by collecting enough linear equations.

13 Ketje

Ketje v2 [BDP+a] 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 It 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 $\mathrm{Init}(K, N) = \mathrm{enc}_8(32)\|K\|10^{119}\|N\|10^*1$ and applies $n_{\mathrm{start}} = 12$ 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 $n_{\mathrm{step}} = 1$-round calls to f, as illustrated in Figure 13, treated with two-bit domain separation flags and 10^*1 padding. The tag is derived using two calls to f, first of which uses $n_{\mathrm{stride}} = 6$ rounds.

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+a], the diffusion between two consecutive sponge states is low. In addition, the algebraic degree of a single round of Keccak$-f$ is only 2. Thus by making queries $(N, A, M_1^i\|0^{r-4})$ with some fixed (N, A) for $i = 1, \ldots, \theta$, we obtain the ciphertexts $C_1^i\|C_2^i$ and the tags, and for each i we can derive

low-degree equations about the bits in the inner (capacity) state just after the processing of N and A (marked red in Figure 13).

Each bit in C_2^i 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 the system and solve it for the state. Once the state is recovered, we can invert the encryption process and recover the key K.

14 Morus

Morus 640 v2 [WHa] (further just Morus) loads the key K and the nonce IV in a state and does some 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 in two steps:

1. $C_i = M_i \oplus S_0 \oplus (S_1 \lll 96) \oplus (S_2 \& S_3)$
2. $(S_0, \ldots, S_4) \leftarrow \mathsf{StateUpdate}(S_0, \ldots, S_4, M_i)$

Nonce-misuse universal forgery and CPA decryption. If we encrypt several messages M^i using the same N, A (in nonce-misuse mode), we can easily see that C_1^i is a function of M_0^i with algebraic degree two. More precisely,when encrypting two messages M^1, M^j with $M_0^j = M_0^1 \oplus \delta$, we can see that

$$\begin{aligned}(C_1^1 \oplus M_1^1) \oplus (C_1^j \oplus M_1^j) =& (\mathsf{Rotl}(\delta, b_1) \lll (w_3 + 96)) \oplus S_2 \& \mathsf{Rotl}(\delta \oplus \mathsf{Rotl}(\delta, b_1), b_3) \oplus S_3 \& (\mathsf{Rotl}(\delta, b_2) \lll w_4) \\ & \oplus (\mathsf{Rotl}(\delta, b_2) \lll w_4) \& \mathsf{Rotl}(\delta \oplus \mathsf{Rotl}(\delta, b_1), b_3)\end{aligned}$$

where Rotl is a linear function, $\lll$ denotes a circular rotation, and all b_r-s and w_t-s are constants. So, with about two different δ values (so four encryption queries), we recover the values of S_2 and S_3 by solving the given linear equation.

Then, once S_2 and S_3 are known for N, A, M^1, $C_1^1 \oplus M_1^1$ can be expressed as a linear function of S_0 and S_1 and we learn their xor-difference. Now we need to recover S_0, S_1, S_4 (also for N, A, M^1), i.e. 384 bits and have 128 linear equations (so 256 unknown bits). To obtain more equations, we make further queries $N, A, \bar{M}^j$ with $\bar{M}^j = M_0^1 \| \bar{M}_1^j \| 0^{128}$. For each query, the value $\bar{C}_2^j$ will supply 128 equations in S_0, S_1, S_4. These will have degree at most 3, however by examining the $\mathsf{StateUpdate}$ 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 making 16 queries in this phase, we should be able to linearise the system and recover S_0, S_1 and S_4 with high probability. Then, the $\mathsf{StateUpdate}$ function can be inverted until we obtain the state after initialization.

Using the state after initialization with IV, we can forge ciphertexts with this IV and decrypt any ciphertext using this IV. So, we have a universal forgery and decryption attack with nonce-misuse and only 20 encryption queries.

15 NORX v3.0

NORX [AJN] is an online cipher 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 $|S| = 128$.

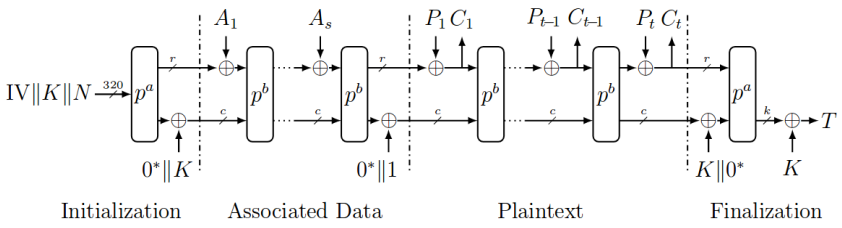

Figure 2: The encryption algorithm of Ascon [DEMS]. Here P is the plaintext.

CPA decryption attack (tiny complexity, nonce misuse.) To decrypt a ciphertext (N, A, C, T), we make $|C|_{384}$ nonce misusing queries. We set $M = \varepsilon$. Then for $i = 1, \ldots, |C|_{384}$ we make an encryption query $(N, A, M\|0^{384})$, obtaining C^i and the tag, and we set $M = M\|(C_i \oplus C_i^i)$. At the end, M will be the plaintext of C. This is because sponge-based schemes XOR each message block with a key stream block computed as a function of N,A, and the previous message blocks.

Semi-universal forgery attack (reusable, nonce-misuse). We can do a nonce-misuse semi-universal forgery attack as follows. It is semi-universal in the sense that we have no control of the nonce in the forgery but we can freely set the rest. Assume we want to forge the encryption of (A, M, Z) with any nonce. For many (about 2^{64}) arbitrary nonces N^i, make an encryption query on $(N^i, A, 0^{384})$and get C_1^i and the tag. Then make encryption queries $(N^i, A, C_1^i\|0^{384})$ (hence a nonce-misuse) so that the state after processing the first block C_1^i will always be $(0, S)$ and get $(0\|C_2^i, T^i)$. Eventually, we will get a collision on S which will induce a collision on (C_2, T) which will thus be visible. With the inputs (N^i, A, C_1^i) and (N^j, A, C_1^j) producing the collision, we can make a forgery on (N^j, A, M, Z) by making an encryption query on the message $(N^i, A, (M_1 \oplus C_1^i \oplus C_1^j)\|M_2\| \cdots \|M_\ell, Z)$, where $M_1, \ldots, M_\ell$ is the sequence of blocks of M. The obtained ciphertext C will yield a valid forgery (N^j, A, C, Z).

This is a semi-universal (there is no choice on N^j, but the choice of (A, M, Z) is free) forgery attack which can be reused with the same (N^j, A). The complexity is the one of finding a collision on the inner state S which is of 128 bits for the instance NORX32-4-1 (32-bit words, 4 rounds, no parallelization, 128-bit tags, 128-bit keys, 128-bit nonces) with a 128-bit security goal [AJN]. So, it is a birthday bound attack.

16 Ascon

Ascon-128 v1.2 [DEMS] is a sponge-based mode for an iterated cryptographic permutation $p : \{0,1\}^{320} \to \{0,1\}^{320}$ 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 16.

CPA decryption attack (tiny complexity, nonce misuse.) As Ascon is a sponge based construction, the same decryption attack as described for NORX can be applied here: to decrypt a ciphertext of ℓ blocks, we need ℓ encryption queries.

Semi-universal forgery (beyond birthday bound, nonce-misusing). We can mount a similar semi-universal forgery attack as for NORX. I.e. to forge a ciphertext for A, M, we do 2^{128} queries

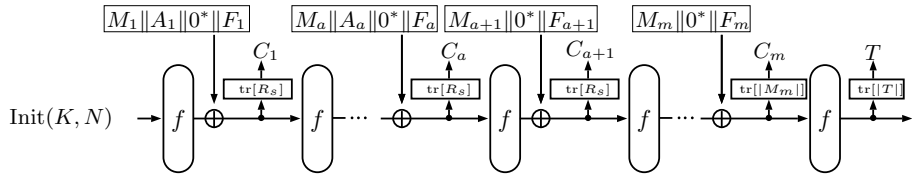

Figure 3: The encryption algorithm of Keyak [BDP+b] when A is processed before M is. We let $tr[x]$ denote truncation to x leftmost (outer) bits.

with distinct nonces N^i[4], A as AD and 0^{64} as message, then redo the queries with the same nonces but use C_1^i as messages to detect a collision on the inner c bits of the state. The collision is then used to create forgeries with A for any M. (N^j, A). Because of Ascon's capacity of 256 bits, the encryption query complexity of finding such a collision is well beyond the birthday bound, and there are simpler generic attacks with the same time complexity.

Universal forgery (nonce-respecting, beyond birthday bound computational complexity). We can a single query (N, ε, M) with arbitrary N and M such that $|M|_{64} = 2$ and with the obtained ciphertext and tag C, T as a witness, we perform an exhaustive search for the secret key. The computational complexity of this attacks is $3 \cdot 2^{128}$ evaluations of the permutation p, which is similar order of magnitude as for the collision-based forgery attack (as every query must be prepared and the result read and stored). However the data and memory complexity of this attack are tiny, compared to the forgery.

17 Keyak

Keyak v2.2 [BDP+b] 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 $R_s = 1344$ 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 sets the state to the value $\mathrm{Init}(K, N) = \mathrm{enc}_8(40)\|K\|10^{183}\|N\|\mathrm{enc}_8(1)\|\mathrm{enc}_8(0)\|F_0$ 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 17. 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.

Semi-universal forgery (reusable, beyond birthday bound, nonce-misusing). We mount a semi-universal forgery (we cannot pick N). We make encryption queries $(N^i, \varepsilon, 0^{Rs})$ obtaining C_1^i and the tag for $i = 0, \ldots, \theta \cdot 2^1 28$. We then make (nonce-misusing) encryption queries $(N^i, \varepsilon, C_1^i\|0^{Rs})$

[4]We note that this exhausts all possible nonces

obtaining $0^{R_s}\|C_2^i, T^i$ for each i. If for some $i \neq i'$ we have $C_2^i, T^i = C_2^{i'}, T^{i'}$, we deduce that $f(\mathrm{Init}(K, N^i)) \oplus C^i\|0^{b-R_s} = f(\mathrm{Init}(K, N^{i'})) \oplus C^{i'}\|0^{b-R_s}$, i.e. that the two nonces yield an inner state collision. To make a forgery for $A, M = M_1\| \ldots \|M_m$ with $N^{i'}$, we obtain the ciphertext and the tag by querying $(N^i, A, (M_1 \oplus C_1^i \oplus C_1^{i'})\| \ldots \|M_m)$.

This is a semi-universal forgery that is reusable for any pair (A, M). Because Keyak uses a capacity of 256 bits, however, the encryption query complexity is well beyond the birthday bound.

CPA decryption attack (tiny complexity, nonce misuse.) We can mount an attack similar to those on Ascon is NORX. As before, to decrypt a ciphertext (N, A, C, T) of $|C|_{1344} = \ell$ blocks, we make ℓ encryption queries $(N, A, M^i\|0^{1344})$ obtaining C^i where $M^i = M_1\| \ldots \|M_{i-1}$ and $M_i = C_i \oplus C_i^i$.

Universal forgery (nonce-respecting, beyond birthday bound computational complexity). As for Ascon, we can perform the exhaustive key search with similar computational complexity as that of the collision-based forgery, using a single encryption query (N, ε, M) with arbitrary N and M such that $|M|_{1344} = 1$. With the obtained ciphertext and tag C, T as a witness, we perform an the exhaustive search with a computational complexity of $\cdot 2^{128}$ evaluations of the permutation f.

18 COLM v1

COLM [ABD+] first derives a secret L from the secret key L by $L = E_K(0)$, where E is AES-128 [DR02]. Again, we use the (bad) 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 just consider that the adversary can choose a sequence M respecting this condition. We let $a = |A|_{128}$ and $\ell + 1 = |M|_{128}$ (M already contains the redundant final block).

First, COLM computes sequences AA and MM by

$$\begin{aligned} \mathsf{AA}_0 &= (N\|\mathsf{param}) \oplus 3 \cdot L & \qquad \mathsf{MM}_i &= M_i \oplus 2^i \cdot L \quad (i = 1, \ldots, \ell - 1) \\ \mathsf{AA}_i &= A_i \oplus 3 \cdot 2^i \cdot L \quad (i = 1, \ldots, a) & \qquad \mathsf{MM}_\ell &= M_\ell \oplus 7 \cdot 2^{\ell-1} \cdot L \\ & & \qquad \mathsf{MM}_{\ell+1} &= M_{\ell+1} \oplus 7 \cdot 2^\ell \cdot L \end{aligned}$$

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,

$$\begin{aligned} C_i &= \mathsf{CC}_i \oplus 3^2 \cdot 2^i \cdot L \quad (i = 1, \ldots, \ell - 1) & \qquad C_\ell &= \mathsf{CC}_\ell \oplus 3^2 \cdot 7 \cdot 2^{\ell-1} \cdot L \\ & & \qquad C_{\ell+1} &= \mathsf{CC}_{\ell+1} \oplus 3^2 \cdot 7 \cdot 2^\ell \cdot L \end{aligned}$$

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 birthday bound complexity. We conclude with an easy nonce-respecting existential forgery attack as the birthday bound complexity.

CCA decryption attack (tiny complexity, using L). Assume we want to decrypt (N, A, C) where A has at least two blocks. We can form A' making a collision on IV with $A'_1 = A_1 + 3 \cdot 6 \cdot L$ and

$A'_2 = A_2 + 3 \cdot 6 \cdot L$ and same other blocks. We obtain

$$\mathsf{AA}'_1 = A'_1 + 3 \cdot 2 \cdot L = A_1 + 3 \cdot 4 \cdot L = AA_2$$

and

$$\mathsf{AA}'_2 = A'_2 + 3 \cdot 4 \cdot L = A_2 + 3 \cdot 2 \cdot L = AA_1$$

So, this choice only permutes AA_1 and AA_2. Hence, it produces the same IV. We can then make decryption query on (N, A', C), and deduce the decryption of (N, A, C).

The total query complexity is of a single decryption query. This is negligible compared to the complexity of getting L.

Universal forgery (tiny complexity, using L). Given (N, A, M), we want to forge the encryption C. First, it is obvious that with access to an E_K oracle, knowing L is enough to simulate the encryption and forge C. So, we will just show how to implement an E_K oracle by having access to a (nonce-respecting) COLM encryption oracle.

We first run a precomputation which will give a set of 128 (x_i, y_i) pairs satisfying $y_i = E_K(x_i)$ and with linearly independent y_i values. For this, we pick a random N, and set $A = A_1$ such that $\mathsf{AA}_1 = \mathsf{AA}_0$. We compute an M for which MM consists of $135 = 128 + \log_2(128)$ zero blocks. Hence, $X_i = L$ for $i = 1, \ldots, 135$. The sequence of chaining values computed for this query, starting with IV, is

$$0, L, 3 \cdot L, 7 \cdot L, 15 \cdot L, \ldots$$

So, $Y_i = L + 3 \cdot (2^{i-1} - 1) \cdot L = 2^{i-1} \cdot L$, and since the representation of $\mathsf{GF}(2^{128})$ is chosen such that 2 is a primitive root, each Y_i for $i = 1, \ldots, 135$ will be distinct. The encryption query (N, A, M) gives C from which we compute CC. With probability close to 1, we will find 128 linearly independent values $y_1, ldots, y_{128}$ among $\mathsf{CC}_1, \ldots, \mathsf{CC}_{135}$. For each y_i, we find j such that $y_i = \mathsf{CC}_j$ and set $x_i = Y_j$, obtaining 128 pairs (x_i, y_i) satisfying $y_i = E_K(x_i)$ and with linearly independent y_i values.

Given the obtained list, whenever we want to compute $E_K(x)$ for an arbitrary x, we first find a set $S \subseteq \{1, \ldots, 128\}$ such that $x + L = 3 \sum_{i \in S} y_i$ (by Gaussian elimination). Then, we compute A_{j+1} so that $\mathsf{AA}_{j+1} = x_{s_j}$ for $j = 1, \ldots, |S|$, and we pick a random N and A_1 so that $\mathsf{AA}_1 = \mathsf{AA}_0$.

With this choice of N, A, we will force $\mathsf{IV} = \sum_{i \in S} y_i$. Then, we set $M_1 = 2 \cdot L$ so that $\mathsf{MM}_1 = 0$ and pick other M_i at random. Thus, we have $X_1 = L$, and $Y_1 = X_1 + 3 \cdot \mathsf{IV} = x$. By querying (N, A, M), we obtain C. We compute CC and deduce $\mathsf{CC}_1 = E_K(x)$.

The total query complexity is really small. There is one encryption query for precomputation. 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 every instance of COLM. We take a constant message M of one block (so C will have two blocks). We take a random block B with last 64 bits matching the guess. We prepare $2^{\frac{128-64}{2}}$ random inputs $(N, \mathsf{param}, A_1, M)$ with unique nonces and $A_1 = (N \| \mathsf{param}) + B$, and we encrypt them using the encryption oracle. Clearly, $\mathsf{AA}_0 + \mathsf{AA}_1 = B + 3^2 \cdot L$. If the guess is correct, this means that 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 two entries such that $\mathsf{AA}'_0 = \mathsf{AA}_1$. So,

$$(N', \mathsf{param}') + 3 \cdot L = \mathsf{AA}'_0 = \mathsf{AA}_1 = (N \| \mathsf{param}) + B + 3 \cdot 2 \cdot L$$

Hence,

$$\mathsf{AA}_0 = (N \| \mathsf{param}) + 3 \cdot L = (N', \mathsf{param}') + B + 3 \cdot 2 \cdot L = \mathsf{AA}'_1$$

This means that the two entries swap AA_0 and AA_1, so produce the same IV. But this collision on IV induce a collision on C, so it is visible. Once we isolate this collision, we deduce the relation $\mathsf{AA}'_0 = \mathsf{AA}_1$ which gives

$$3^2 \cdot L = (N \| \mathsf{param}) + (N', \mathsf{param}') + B$$

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 make 2^{128-2m} batches of 2^m queries. In the i^{th} batch, we query $(N^i, A^{i,j}, M)$ for $j = 1 \ldots, 2^m$ with N^i fixed in the batch, M fixed globally, and random $A^{i,j}$ of 2 blocks. We then find i and $j \neq j'$ such that $C^{i,j}, T^{i,j} = C^{i,j'}, T^{i,j'}$, and recover L from $A_1^{i,j}$ and $A_1^{i,j'}$ similarly as before. We note that when $m = 64$, this attack has birthday complexity.

Semi-universal forgery (nonce misuse, birthday bound). In this attack, we mount a semi-universal forgery (we can only choose M) at the birthday bound complexity. We pick a constant M and for $i = 1, \ldots, 2^{64}$ query (N^i, A^i, M) until we obtain a collision on C^i, T^i. We get (N^i, A^i) and (N^j, A^j) for which we deduce, from the construction, we deduce that there is an inner collision on IV. As IV only depends on the nonce and AD, we can forge a ciphertext for any $\bar{M}$ by querying $(N^i, A^i, \bar{M})$ to get $\bar{C}, \bar{T}$ and forging with $(N^j, A^j, \bar{C}, \bar{T})$. Note that every such forgery must misuse nonce, as the collision search exhausts all nonces.

Existential forgery. We conclude with a nonce-respecting existential forgery attack at the birthday bound complexity. We pick a constant M_1 and encrypt many (N, A_1, M_1, M_2) until we obtain a collision on C_1. We get (N, A_1, M_1, M_2) and (N', A_1', M_1, M_2') producing (C_1, C_2, C_3) and (C_1, C_2', C_3'), respectively. From the construction, we deduce that there is an inner collision on IV. As IV only depends on the nonce and A, we deduce that C' is a forgery for the encryption of (N, A_1, M_1, M_2').

19 Jambu

Jambu v2.1 [WHb] 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 computing $R_{a+i}\|U_{a+i}\|V_{a+i} \leftarrow F_K(R_{a+i-1}\|U_{a+i-1}\|V_{a+i-1}, 0, M_i)$ 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 last fractional block of plaintext M_m is processed (if $|M| \bmod \nu = 0$, we set $M_m = \varepsilon$) by computing $R_{a+m}\|U_{a+m}\|V_{a+m} \leftarrow F_K(R_{a+m-1}\|U_{a+m-1}\|V_{a+m-1}, 0, M_m\|10^*)$ and $C_m \leftarrow \text{trunc}_{|M_m|}(M_m\|10^* \oplus V_{a+m})$.

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 $T_1\|T_2 \leftarrow E_K(U_{a+m+1}\|V_{a+m+1})$.

Universal forgery (number of decryption queries at birthday-bound, nonce-respecting). Because the tags of Jambu are only 64-bit long, the trivial tag guessing attack only requires 2^{642} decryption queries. We can make a forgery for any target triplet (N, A, M) with a single encryption query, but requiring many decryption queries.

To forge a ciphertext for (N, A, M) with $|M|_{64} = m$ (and with $1 \leq |M_m| < \nu$), pick an arbitrary message $M' = M_1\| \ldots \|M_{m-1}\|M_m'$ with $\Delta = M_m\|10^* \oplus M_m'\|10^* \neq 0^\nu$ and $|M_m'| \geq |M_m|$. We make an encryption query with (N, A, M') to obtain C', T' and set $C = C_1'\| \ldots \|C_{m-1}'\|C_m$ with

$C_m = \text{trunc}_{|M_m|}(C'_m) \oplus \text{trunc}_{|M_m|}(\Delta)$. Then, we try to forge with (N, A, C, T) for all possible values $T \in \{0,1\}^\nu$ until we succeed. .

Semi-universal forgery (beyond birthday-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 forge a ciphertext for a target plaintext M with $O(2^{96})$ encryption queries, however we cannot choose the AD and the attack is nonce-misusing

We first pick a nonce N and a message M' of $128u$ bits and make encryption queries (N, A^j, M') obtaining C^j, T^j for $j = 1, \ldots, \theta \cdot 2^{3n/4}$ such that for every j the AD A^j is a random $(2\nu - 1)$-bit string different from $A^1, \ldots, A^{j-1}$. We then find $1 \leq j < j' \leq 2^{96}$ for which $C^j, T^j = C^{j'}, T^{j'}$. We deduce that in these queries, the internal states just after AD processing must collide.We then make an encryption query with (N, A^j, M), obtaining C, T and forge with $(N, A^{j'}, C, T)$.

Existential forgery (beyond birthday-bound, nonce-respecting). The too-few-nonces problem of the previous attack can be circumvented if we fold many subqueries into every encryption query.

We pick a ν-bit constant string P. We then make encryption queries (N^j, A^j, M^j) obtaining C^j, T^j for $j = 1, \ldots, \theta \cdot 2^{48}$ such that for every j, the nonce is fresh, the AD A^j is a random $(\nu - 1)$-bit string, and the message $M^j = \|_{k=1}^{\theta \cdot 2^{48}} Q_k^j \| P \| P$ where every Q_k^j is a random ν bit string. We then find $(j, i) \neq (j', i')$ such that $i \equiv i' \equiv 2 \pmod 3$ and $C_i^j \| C_{i+1}^j = C_{i'}^{j'} \| C_{i'}^{j'}$. We deduce that the j-th and j'-th queries had an internal state-collision $R_i^j \| U_i^j \| V_i^j = R_{i'}^{j'} \| U_{i'}^{j'} \| V_{i'}^{j'}$. We can then forge with $(N^j, A^j, C_1^j \| \ldots \| C_{i-1}^j \| C_{i'}^{j'} \| \ldots \| C_{\theta \cdot 2^{3n/8}}^{j'}, T^{j'})$.

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. Also, this strategy cannot be used for a universal forgery.

References

[ABD+] 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.

[AJN] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX v3.0. https://competitions.cr.yp.to/round3/norxv30.pdf.

[BDP+a] Guido Bertoni, Joan Daemen, MichaÃńl Peeters, Gilles Van Assche, and Ronny Van Keer. CAESAR submisssion: Ketje v2. https://competitions.cr.yp.to/round3/ketjev2.pdf.

[BDP+b] Guido Bertoni, Joan Daemen, MichaÃńl Peeters, Gilles Van Assche, and Ronny Van Keer. CAESAR submisssion: Keyak v2. https://competitions.cr.yp.to/round3/keyakv22.pdf.

[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.

[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.

[Bera] D. J. Bernstein. Cryptographic competitions: CAESAR. "https://competitions.cr.yp.to/caesar-call.html".

[Berb] D. J. Bernstein. Cryptographic competitions: Disasters. https://competitions.cr.yp.to/disasters.html.

[BR00] Mihir Bellare and Phillip Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In *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.

[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.

[CAE] Cryptographic competitions: CAESAR submissions. "http://competitions.cr.yp.to/caesar-submissions.html".

[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.

[DEMS] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schl affer. Ascon v1.2. https://competitions.cr.yp.to/round3/asconv12.pdf.

[DMA17] 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.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[Fer02] N Ferguson. Collision attacks on ocb. *NIST CSRC website*, 2002.

[Fer05] Niels Ferguson. Authentication weaknesses in gcm. 2005.

[FFL12] Ewan Fleischmann, Christian Forler, and Stefan Lucks. Mcoe: A family of almost foolproof on-line authenticated encryption schemes. In *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.

[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.

[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.

[HKR] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. AEZ v5: Authenticated encryption by enciphering. https://competitions.cr.yp.to/round3/aezv5.pdf.

[HKR15] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. Robust authenticated-encryption AEZ and the problem that it solves. In *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.

[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.

[HW] Tao Huang and Hongjun Wu. Attack on AES-OTR. https://groups.google.com/forum/#!topic/crypto-competitions/upaRX2jdVCQ.

[IMG+] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, Sumio Morioka, and Eita Kobayashi. CLOC and SILC. https://competitions.cr.yp.to/round3/clocsilcv3.pdf.

[IOM12] Tetsu Iwata, Keisuke Ohashi, and Kazuhiko Minematsu. Breaking and repairing GCM security proofs. In *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.

[JNP] JÃľrÃľmy Jean, Ivica NikoliÄĞ, and Thomas Peyrin. Deoxys v1.41. https://competitions.cr.yp.to/round3/deoxysv141.pdf.

[Jon03] Jakob Jonsson. On the security of CTR + CBC-MAC. In *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002. Revised Papers*, volume 2595 of *Lecture Notes in Computer Science*, pages 76–93. Springer, 2003.

[Jou06] Antoine Joux. Authentication failures in nist version of gcm. 2006.

[KR] Ted Krovetz and Phillip Rogaway. OCB (v1.1). https://competitions.cr.yp.to/round3/ocbv11.pdf.

[KY01] Jonathan Katz and Moti Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In *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, 2001.

[Lu17] Jiqiang Lu. Almost universal forgery attacks on the copa and marble authenticated encryption algorithms. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 789–799. ACM, 2017.

[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.

[Min] Kazuhiko Minematsu. AES-OTR v3.1. https://competitions.cr.yp.to/round3/aesotrv31.pdf.

[MV04] David A. McGrew and John Viega. The security and performance of the galois/counter mode (GCM) of operation. In *Progress in Cryptology - INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, December 20-22, 2004, Proceedings*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004.

[Nik] Ivica NikoliÄĞ. Tiaoxin - 346. https://competitions.cr.yp.to/round3/tiaoxinv21.pdf.

[PC15] Gordon Procter and Carlos Cid. On weak keys and forgery attacks against polynomial-based MAC schemes. *J. Cryptology*, 28(4):769–795, 2015.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 98–107, 2002.

[RS06] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In *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.

[RW03] Phillip Rogaway and David A. Wagner. A critique of CCM. *IACR Cryptology ePrint Archive*, 2003:70, 2003.

[Saa12] Markku-Juhani Olavi Saarinen. Cycling attacks on gcm, GHASH and other polynomial macs and hashes. In *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.

[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.

[WFH03] Doug Whiting, Niels Ferguson, and Russell Housley. Counter with cbc-mac (ccm). 2003.

[WHa] Hongjun Wu and Tao Huang. The authenticated cipher MORUS (v2). https://competitions.cr.yp.to/round3/morusv2.pdf.

[WHb] Hongjun Wu and Tao Huang. The JAMBU lightweight authentication encryption mode (v2.1). https://competitions.cr.yp.to/round3/jambuv21.pdf.

[WP] Hongjun Wu and Bart Preneel. AEGIS: A fast authenticated encryption algorithm (v1.1). https://competitions.cr.yp.to/round3/aegisv11.pdf.

[Wu] Hongjun Wu. ACORN: A lightweight authenticated cipher (v3). https://competitions.cr.yp.to/round2/acornv2.pdf.