



# Chip-to-Chip Authentication Method Based on SRAM PUF and Public Key Cryptography

Ioannis Karageorgos<sup>1,2</sup> · Mehmet M. Isgenc<sup>1</sup> · Samuel Pagliarini<sup>1</sup> · Larry Pileggi<sup>1</sup>

Received: 17 January 2019 / Accepted: 27 September 2019 / Published online: 26 November 2019  
© The Author(s) 2019

## Abstract

In today's globalized integrated circuit (IC) ecosystem, untrusted foundries are often procured to build critical systems since they offer state-of-the-art silicon with the best performance available. On the other hand, ICs that originate from trusted fabrication cannot match the same performance level since trusted fabrication is often available on legacy nodes. Split-Chip is a dual-IC approach that leverages the performance of an untrusted IC and combines it with the guarantees of a trusted IC. In this paper, we provide a framework for chip-to-chip authentication that can further improve a Split-Chip system by protecting it from attacks that are unique to Split-Chip. A hardware implementation that utilizes an SRAM-based PUF as an identifier and public key cryptography for handshake is discussed. Circuit characteristics are provided, where the trusted IC is designed in a 28-nm CMOS technology and the untrusted IC is designed in an also commercial 16-nm CMOS technology. Most importantly, our solution does not require a processor for performing any of the handshake or cryptography tasks, thus being not susceptible to software vulnerabilities and exploits.

**Keywords** Chip-to-chip authentication · Split-Chip · PUF · SRAM · Public key cryptography · RSA · ASIC

## 1 Introduction

The increased complexity in integrated circuit (IC) manufacturing processes entails an ever increasing capital investment to establish an IC foundry. Companies and governments that once used to have full oversight over design and fabrication of their critical systems, now commonly delegate the fabrication to third parties. In order to keep at pace with the increased complexity and cost of IC manufacturing, foundries have been establishing themselves in regions that are cost favorable. In practice, the state-of-the-art foundries of today are considered untrusted entities in the IC supply

chain, which raises some serious considerations about the trustworthiness of the fabricated ICs [1–3].

Relying on untrusted fabrication requires a careful assessment of its threats. For instance, any IP that is part of the system is inevitably shared with the untrusted foundry. Besides the obvious risks of IP theft, reverse engineering (from design database), and overproduction, the same IP can also be modified through the insertion of a backdoor or a hardware trojan horse (HTH). For a thorough discussion on HTH types and activation conditions, please see [4].

There are, however, alternatives to combat the threats of untrusted fabrication. Trojan detection, despite several shortcomings, has active research efforts [4]. Obfuscation [5] and logic locking [6] are effective countermeasures to a certain extent, with several variants described in the literature. Generally speaking, these circuit-level strategies require modifications to the circuit that would render an adversary less capable of making sense of the IP.

Trusted fabrication is another option to protect IP from untrusted parties. While trusted fabrication is reliable from a hardware security standpoint, there are financial and logistical obstacles to it. Maintaining a trusted fabrication facility is prohibitively expensive, specially at advanced nodes.

---

**Electronic Supplementary Material** The online version of this article (<https://doi.org/10.1007/s41635-019-00080-y>) contains supplementary material, which is available to authorized users.

✉ Ioannis Karageorgos  
ioannis.karageorgos@yale.edu

<sup>1</sup> Department of Electrical and Computer Engineering, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA

<sup>2</sup> Present address: Department of Electrical Engineering, Yale University, 10 Hillhouse Ave, New Haven, CT 06511, USA

In recent years, several research efforts relating to hybrid fabrication approaches have been made. In particular, IARPA’s TIC program was a catalyst for research into split manufacturing [7], an approach where a single IC is built by both an untrusted and a trusted foundry. The untrusted foundry is responsible for the lower layers (transistors and M1, typically) while the trusted foundry is responsible for the upper metal layers. The appeal of the approach is that the performance of the state-of-the-art transistors can be leveraged without revealing the entire layout of the IC to the untrusted foundry, thus protecting any sensitive IP.

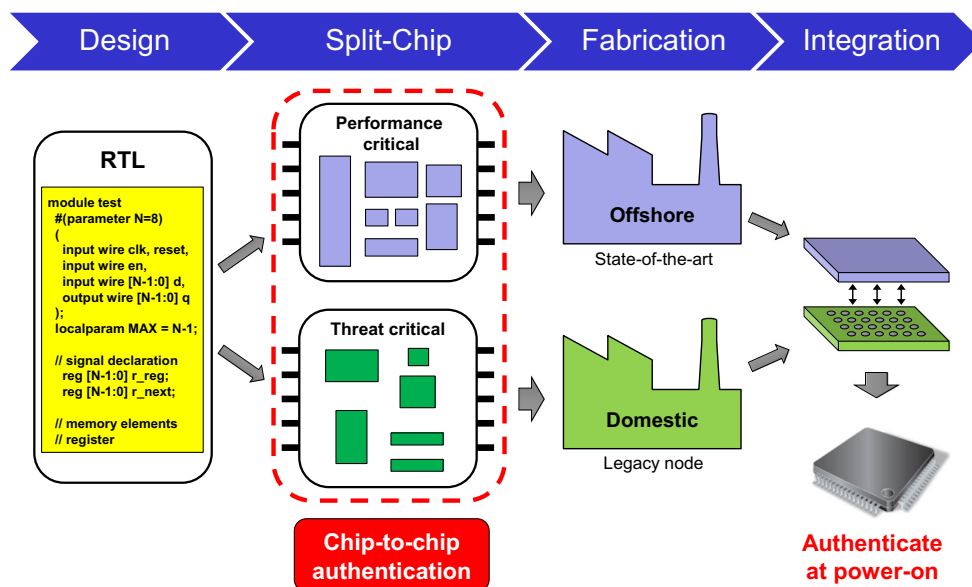
The background technique for this work is also a hybrid fabrication approach, which we refer to as Split-Chip [8]. In Split-Chip, the goal is to build a trustworthy system composed of two ICs, one trusted and one untrusted. Similar to split manufacturing, the performance of the untrusted technology can be leveraged while critical IP is not shared. An illustration of this hybrid fabrication approach is shown in Fig. 1. The flow starts by coding RTL for the entire system—the code at this stage is completely agnostic to Split-Chip. The system is then partitioned into two ICs, a process that is driven by the performance and criticality constraints of the individual modules. Finally, both ICs are fabricated and later integrated to rebuild the system intent.

Split-Chip brings unique attacks that are specific to its dual-IC framework. In this work, we present a method to securely authenticate the untrusted and trusted ICs and drastically reduce risks associated with potential malicious acts carried out by an attacker. Although our authentication method has been originally developed for use in a Split-Chip context, in practice, it can be applied to any system that requires chip-to-chip authentication.

The remainder of this paper is organized as follows: Section 2 provides the background on the Split-Chip technique, as well as a description of adversaries and threats. In Section 3, we present our authentication method along with the detailed architecture of the system. In Section 4, we show implementation results based on commercial CMOS technologies. We then proceed by comparing our results with other solutions in the literature, both software-based and hardware-based. Finally, Section 5 concludes our paper and provides some directions for future research.

## 2 Background on Split-Chip

As previously mentioned, the key characteristic of Split-Chip is that a trustworthy system is built by combining two ICs. There are unique challenges related to the design and partition of such systems. Generally speaking, IP that is considered critical should be placed on the trusted portion of the system. Non-critical IP can reside on the trusted or on the untrusted ICs, with different implications for the system performance. The technologies selected for the trusted and untrusted ICs can be drastically different, such that assigning modules to a given IC requires careful assessment. Assuming the trusted IC is fabricated in a legacy CMOS node, the need for an automated assessment and partition tool is exacerbated. Such partitioning tool is discussed at length in [9]. The integration and packaging choices for a Split-Chip system also have a heavy bearing on the system performance. Inevitably, an interface between the two ICs will be established for exchanging data. It is foreseeable that the ICs will exchange data that could



**Fig. 1** Illustration of the Split-Chip fabrication flow. The proposed authentication scheme is implemented during the design phase of the “splitting” process. The authentication scheme is started at power-on

otherwise expose sensitive information, for which a secure data exchange mechanism is also required. Therefore, we utilize authentication and encryption approaches to realize a secure chip-to-chip interface.

While split manufacturing [7] is widely considered an effective approach, it compulsorily requires the untrusted foundry to deliver unfinished wafers. Commercial foundries are not easily convinced to change their flow and pull unfinished wafers. To a certain extent, the logistical barrier to implementing split manufacturing is higher than the technological barrier (i.e., alignment). The most significant motivation to pursue a Split-Chip solution is that it does not require any degree of cooperation from a foundry. In fact, the untrusted foundry is not even capable of realizing it has been procured to build a portion of a Split-Chip system.

## 2.1 Adversaries and Attack Models

A detailed list of attacks and respective adversaries is provided in Table 1. The threats that are specifically addressed by this work are highlighted. Notice how the trusted foundry is never considered an adversary as it is assumed complete oversight of the trusted fabrication is in place. Therefore, the adversaries that we consider are the untrusted foundry and a reverse engineer. It is assumed that the untrusted foundry has unrestricted access to the layout/netlist of the untrusted IC, while the reverse engineer can obtain the layout/netlist of both ICs through imaging and delayering techniques of fabricated chips. We list four different attacks that the untrusted foundry could mount, yet all of them are thwarted by the fact that the untrusted foundry has only partial understanding of the system as a whole.

Attack scenarios AS1 and AS2 relate to the removal or insertion of malicious logic. As is the case with other hybrid fabrication approaches, Split-Chip cannot deter the untrusted foundry from performing HTH insertion. However, the untrusted foundry has only partial overview of the system and would therefore have less opportunities for targeted HTH insertion. For instance, if the authentication logic is modified, chips will not pair and the attack will fail.

On the other hand, AS3 and AS4 relate to theft of IP or the entire IC in the case of overproduction. However, the untrusted foundry cannot produce or steal the portions of the system that it cannot see. As long as the system partitioning is carefully realized, no sensitive IP should be shared with the untrusted foundry.

For a Split-Chip scheme to work, the ICs must exchange keys securely such that an adversary cannot perform an eavesdropping attack by simply monitoring the I/Os of either IC. The use of encryption prevents AS6—and the stronger the encryption, the stronger the mitigation.

Preventing an attacker from mounting AS5 is, in practice, not possible. However, by authenticating and bonding pairs of chips, even if an attacker is able to acquire a pre-authenticated pair of ICs, it still is not possible to learn signatures from other chips. The acquired trusted IC will not pair to any other untrusted IC and vice-versa. Let us now discuss how this functionality is achieved.

## 3 Proposed Method and Implementation

To protect a Split-Chip system from the risks associated with untrusted manufacturing, as denoted in Section 2, we develop a method which relies on a physically unclonable function (PUF). For the specific implementation presented in this paper, the power-up state of an SRAM is utilized as the PUF. This is not a restriction, other PUF sources would be equally sufficient but may prove harder to integrate than SRAM. Along with the PUF, public key cryptography is utilized to provide a secure way of authenticating the pair of ICs.

The proposed method and implementation have been optimized such that minimal area and power overheads are present. Besides pairing the chips, the implemented method can be partially reused for any subsequent encrypted communication between them.

### 3.1 SRAM Power-up State as a Fingerprint

The most commonly used practice for IC authentication involves the placement of a secret key in some kind

**Table 1** Attack scenarios and their respective attackers

Scenario	Adversary	Target	Approach	Mitigation strategy
AS1	Untrusted foundry	Untrusted IC	HTH/backdoor insertion	Attacker has only partial understanding of the system
AS2	Untrusted foundry	Untrusted IC	Circuitry removal	Attacker has only partial understanding of the system
AS3	Untrusted foundry	Untrusted IC	Overproduction	Untrusted IC alone does not perform the same system function
AS4	Untrusted foundry	Untrusted IC	IP theft	Critical IP is not shared with the untrusted foundry
AS5	Reverse engineer	Both ICs	Delayering, imaging, etc.	Pairing and authentication scheme (this work)
AS6	Reverse engineer	Interface	Eavesdropping	Encryption (this work)

of non-volatile electrically erasable programmable read-only memory (EEPROM) and the use of cryptographic operations based on that key. The secret key serves as the “fingerprint” or digital signature of the IC. While this approach works well for most consumer applications, it is still susceptible to various attacks from more capable adversaries.

PUFs have emerged as a promising solution for secure secret key storage. A PUF is a mathematical function that describes the complex behavior of a physical structure derived from inevitable physical variations during semiconductor manufacturing. Secret keys can be derived directly from that function, without the requirement of secure non-volatile memories and other expensive hardware. Since the secret key is derived from physical characteristics of the IC, any physical attack attempting to extract subproducts of the PUF from the chip should be performed while the chip is powered on, which is considerably more difficult [10]. Furthermore, invasive attacks are more difficult to execute without modifying the underlying physical characteristics from which the key is derived [11].

A typical SRAM cell contains two cross-coupled inverters which reinforce each other into one of two stable states. Similar to applying a reset pulse to an SR-latch, a write operation forces the transition of the cell towards one of those states. Interestingly, before the first write operation, i.e., during device power up, the cell exists in a metastable state where the force of both feedback loops is theoretically equal and it cannot transition to one of the stable states. In practice, however, one of the inverters is always slightly stronger than the other mainly due to process variation-induced transistor threshold mismatches during fabrication. Thus, each cell “settles” almost always into the same state.

The idea of using the initial state of an SRAM as an electronic fingerprint has been explored [12] and demonstrated in real silicon [13]. Results from [13] show more than 96% bit stability and a nearly ideal Hamming distance distribution between individual chips. These results were obtained with a custom designed SRAM cell, but other works have achieved comparable bit stability levels using off-the-shelf RAM [14]. According to [15], SRAM-based PUF exhibits the best PUF behavior, with high reliability and the largest Hamming distance. Authors in [16] report an average entropy present in SRAM PUF of 950 per 1,000 bits, which is the highest among all other types of tested PUFs.

The above findings, combined with the fact that SRAMs are prevalent in IC design and display rather high density, motivate us to use SRAM-based PUF as the fingerprint mechanism in our authentication method.

### 3.2 Public Key (Asymmetric) Cryptography

RSA is the first practical realization of public key cryptography and has gained extreme popularity since it was first published to the unclassified community [17, 18]. Similar to other public key cryptosystems, RSA is based on the notion of a trapdoor one-way function—a function that is easy to compute in one direction but very difficult (as far as is currently known) to compute in the opposite direction. Particularly, the RSA algorithm exploits the prime factorization problem which refers to the difficulty in factoring the product of two large primes. Indeed, while it is relatively easy to compute the product of two large primes, doing the opposite, i.e., factorize the resulting product into the original prime numbers, is still very difficult.

The RSA algorithm can be split into three parts: the key pair generation, data encryption, and subsequent decryption of data. The key pair generation precedes any encryption or decryption process. The basics of all three parts are as follows. We do not discuss issues related to RSA padding/short messages as our implementation does not give the user the freedom to pick the message length (or content).

#### A. Key pair generation

An  $N$ -bit key pair can be generated using the following steps.

1. Find two random prime numbers  $p, q$  of approximately  $(N/2)$ -bit size, such that their product is of size  $N$
2. Compute  $n = p \times q$  and  $\phi(n) = (p - 1) \times (q - 1)$ , where  $n$  is the modulus and  $\phi(n)$  the Euler’s totient function
3. Choose an integer  $e$ , with  $1 < e < \phi(n)$ , such that  $\text{GCD}(e, \phi(n)) = 1$ , where  $\text{GCD}$  is the greatest common divisor. Integer  $e$  is known as the public exponent and it is usually a prime number and most commonly one of the first Fermat numbers
4. Compute the secret exponent  $d$  as the modular multiplicative inverse of  $e \pmod{\phi(n)}$ . Based on the extended Euclidean algorithm,  $d$  can be computed as  $d \equiv e^{-1} \pmod{\phi(n)}$

The public key is defined as the pair of public exponent and modulus,  $pk = \{e, n\}$ . The secret key is defined as the pair of secret exponent and modulus,  $sk = \{d, n\}$ . Finding the large random primes  $p$  and  $q$  is the most time-consuming step of RSA and it determines, to a large extent, the total time required for the key pair generation.

**B. Encryption**

A plaintext  $m$ , where  $1 < m < n$ , is encrypted to a ciphertext  $c$  by modular exponentiation with the public key as  $c = m^e \bmod n$ .

**C. Decryption**

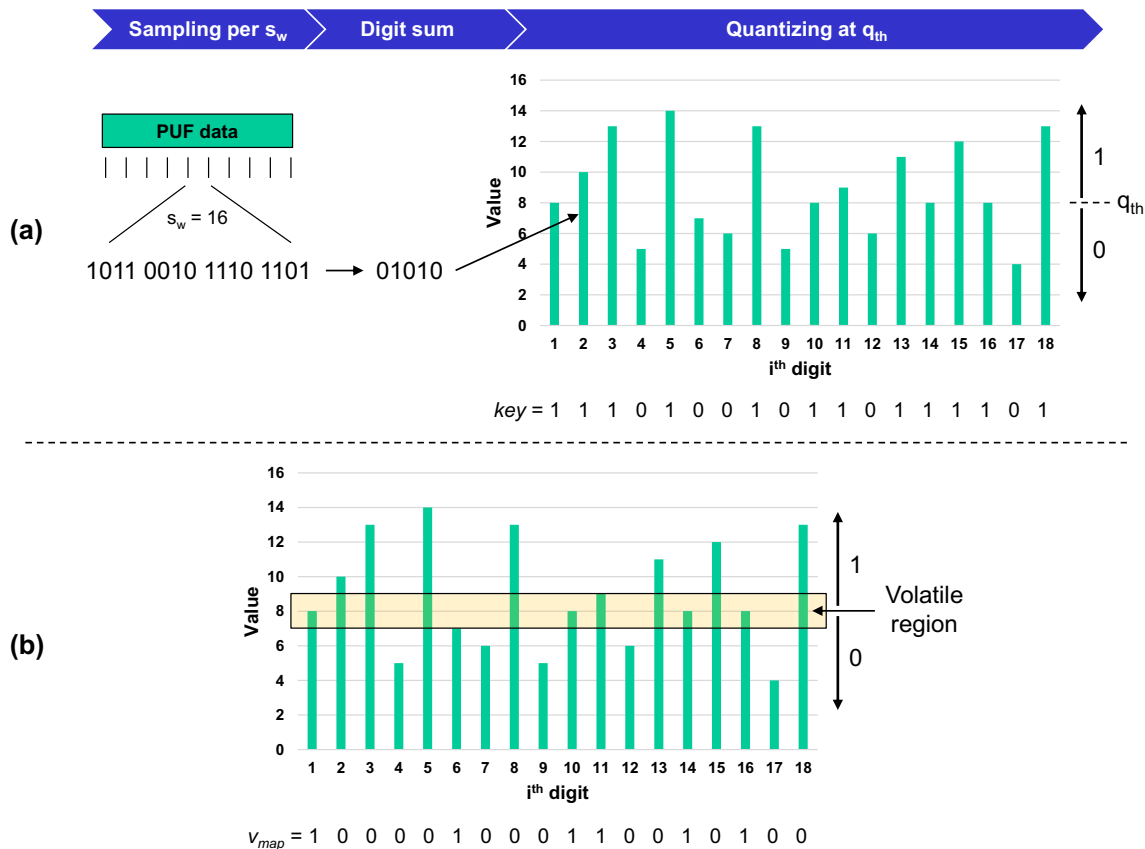
A ciphertext  $c$  is decrypted to the original plaintext  $m$  by modular exponentiation with the secret key as  $m = c^d \bmod n$ .

**3.3 Noise Filtering of Fingerprint**

One of the related challenges of using an SRAM PUF for cryptographic operations is its inherent stochasticity. Indeed, the operation of the PUF relies on physical properties that are naturally susceptible to noise and variability due to environmental factors and aging. When transistor threshold mismatches are small, SRAM cells create sufficiently similar pull-up and pull-down tensions in the feedback loop. In this case, small environmental fluctuations or aging can cause enough of an imbalance such that a bit can flip and no longer settle to its ‘original’ state. However, cryptography relies heavily on reproducibility of secret keys. The output of a cryptographic function, such as

RSA, is extremely sensitive to its input values. Therefore, in order to use the output of an SRAM PUF as the secret key, it needs to be error corrected, or filtered, first.

We propose a simple, fast, and scalable filtering based on the quantization of an excessive amount of SRAM bits to produce a noise-resistant full entropy key. The first step of the filtering process is to define a sampling width ( $s_w$ ). This denotes the number of bits to be quantized later. For example, a sampling width of 8 means that every chunk of 8 bits represents 1 sample. Next, we calculate the digit sum of each sample by counting the ones (“1”) among its digits, i.e., the 8-bit sample of “10110111” will be converted to the decimal 6 (or binary “0110”). This is often referred to as the Hamming weight of the sample. The outcome will be a new bit stream of  $x$  words, where  $x$  is the target key length. The width of each word is equal to  $\lfloor \log_2(s_w) \rfloor + 1$ . We denote this bit stream as  $s$  and each word/sample as  $s_i$ . Then we quantize each sample by setting the threshold ( $q_{th}$ ) at half the sampling width (e.g., for  $s_w = 8 \Rightarrow q_{th} = 4$ ) and we derive a binary key of  $x$  digits. If the value of each sample ( $s_i$ ) is less than the threshold ( $q_{th}$ ), then it will be set to 0. Otherwise, it will be set to 1. We denote this as  $key$ . A schematic representation of the process is shown in Fig. 2a for a 16-bit sample.



**Fig. 2** Schematic representation of our noise filtering method. **a** Sampling, digit sum (Hamming weight) and quantization of the SRAM bitstream. **b** Definition and mapping of the volatile region

While this method can provide a cheap and efficient way to tolerate random bit flips, as a result of environmental or aging factors, it is not enough as a solution. Quantization alone cannot assure reproducibility of a noisy signal due to the uncertainty problem at, or near, the quantization threshold. When the value of  $s_i$  equals  $q_{th}$ , even a single bit flip can change the relevant quantized digit ( $key_i$ ). To overcome this problem, we perform two additional steps to filter out the potential unstable digits. First, we define another parameter which specifies the width of the region we want to filter. We call this region the volatile region and we denote the parameter as  $v_w$ . A  $v_w = 3$  means that there are 3  $s_i$  values which fall into this region with  $q_{th}$  as the middle value. For example, for  $v_w = 3$  and  $q_{th} = 4$ , every  $3 \leq s_i \leq 5$  belongs to the volatile region and will be referred to as a volatile digit. Then the first step is to “map” the positions of volatile digits in the *key*. A new bit stream is created with length equal to the *key* length which is the binary representation of this map. The ones (“1”) in this bit stream represent a volatile digit and the zeros (“0”) represent a non-volatile digit. We denote this as  $v_{map}$ . A schematic representation of the extra filtering process is shown in Fig. 2b.

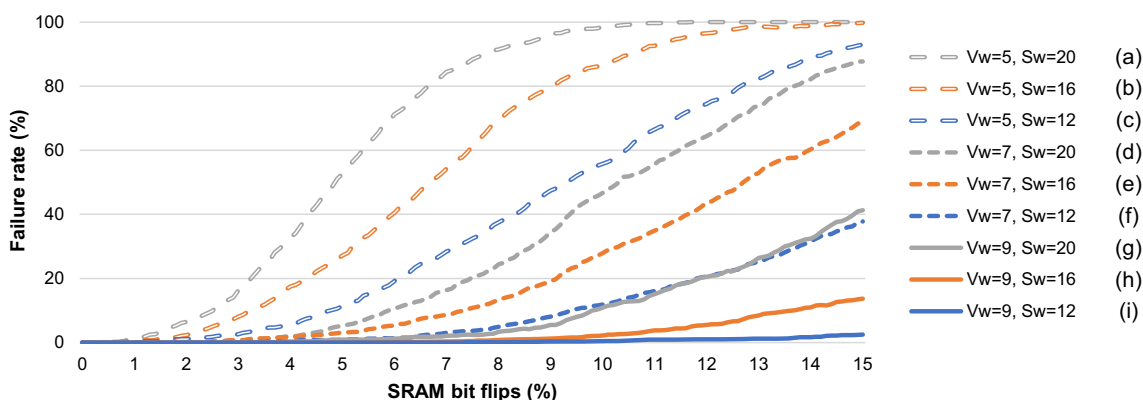
The map of volatile digits is to be written in an one-time programmable (OTP) embedded non-volatile memory (eNVM), such as an anti-fuse, along with an extra bit that will serve as the “write flag”. Namely, the mapping process will run only once during the first IC power-up and the outcome will be saved permanently on the IC. In the final step, we use this map as a “mask” in order to exclude the volatile digits of the key, which are set to a fixed value. Thus, the final/filtered key is defined as  $key_f = key \text{ AND } \overline{v_{map}}$  for a fixed value of “0” for the volatile digits and  $key_f = key \text{ OR } v_{map}$  for a fixed value of “1.” This process reduces the entropy of the derived key which can be compensated by increasing the key length.

The filter’s resistance to noise is ultimately controlled by tuning the sampling width ( $s_w$ ) and volatile region width

( $v_w$ ) parameters. We calculated its effectiveness over a range of  $s_w$  and  $v_w$  by introducing random bit flips on the initial PUF key and measuring the failing rate over 1,000 samples per case. In each sample, a fixed number of bit flips is introduced at random positions. The filter fails over a sample when the resulting quantized key differs from the original quantized key. Our statistical results over a wide range of number of bit flips are plotted in Fig. 3.

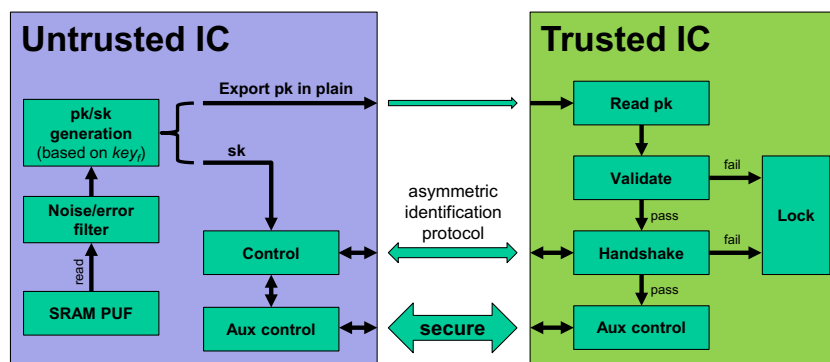
The length of the PUF key is the product of the target key length times the sampling width. The filter’s resistance to noise increases when the sampling width decreases and when the volatile region width increases. Decreasing  $s_w$  means that we allow “more frequencies” in our key which results in increased contrast between samples, and thus increased distance from quantization threshold. If we take a very large sample, it will average very close to a 1-to-1 ratio of zeros and ones (assuming no transistor biasing), i.e., at  $s_w/2$ . Furthermore, since there are more samples, the chance of multiple bit flips inside the same sample reduces. However, decreasing  $s_w$  also reduces the margins outside of the volatile region, which reduces the entropy of the key since a higher percentage of samples would get a value inside the volatile region. Accordingly, the increase of  $v_w$  also reduces the entropy of the key. Thus, choosing the values of  $s_w$  and  $v_w$  is a trade-off between noise resistance, entropy, and PUF size. For example, the cases (f) and (g) in Fig. 3 show very similar response to noise. However, case  $\{v_w = 7, s_w = 12\}$  results in a key with lower entropy but much smaller PUF size compared with case  $\{v_w = 9, s_w = 20\}$  and vice versa. A designer can choose one over the other depending on the design requirements.

In [19], results from extensive experiments on SRAM PUFs found on FPGAs are shown. Multiple measurements of the power-up state of 8,190 bytes of SRAM from different memory blocks on different FPGAs show an average intra-distance (noise level) of 3.57%. The authors in [14] and [20] report an average intra-distance of 3.8% for 5,120



**Fig. 3** Statistical results of the PUF filter and its resistance to noise. Failure rates of the filter (y axis) are plotted against the rate of bit flips in the original PUF key (x axis), for various combinations of  $v_w$  and  $s_w$  values

**Fig. 4** Block diagram of the proposed Split-Chip system implementation



blocks of 64 SRAM cells measured on eight commercial SRAM chips and an average intra-distance of 6.5% for 15 blocks of 64 SRAM cells from the embedded memory in three microcontroller chips. As reported in [19] and [16], noise level increases with accelerated effective aging of 10–20 years if no countermeasures are taken. However, when aging countermeasures (anti-aging mechanism) are introduced, the impact of aging completely vanishes. They also report a strong resiliency to environmental variations, such as temperature, voltage, and electromagnetic radiation with < 12% intra-distance at extreme values. These results indicate that a filter configuration with (e), (f), or (g) ensures high reliability (higher yield) while (b), (c), or (d) can be used when key entropy is prioritized over reliability.

### 3.4 Split-Chip Authentication Solution

We now discuss how asymmetric cryptography can be combined with an SRAM PUF and its filter to form a secure authentication solution that is tailored to Split-Chip. We design and implement the hardware for all three parts. Portions of the design reside on the trusted IC and portions of the design reside on the untrusted IC. The top-level block diagram of the system is shown in Fig. 4.

The proposed authentication solution is based on the following assertions:

- The untrusted IC performs RSA crypto key pair generation as a power-on routine. The secret/private key ( $sk$ ) is kept in memory while the public key ( $pk$ ) is exported in plain.
- After fabrication, the untrusted ICs are put through rigorous testing for physical defects. If trojan detection mechanisms are part of the design, they can also be exercised at this stage.<sup>1</sup>
- For those ICs that pass testing, their public keys are stored and embedded into their trusted IC counterparts.

<sup>1</sup>It is always important to acknowledge that test vectors alone are very inefficient to trigger HTHs. Alternative detection schemes can be devised and would not interfere with the authentication solution herein proposed, but could make it even stronger

Any memory technology can be used for storing the public keys, but write-once memories like fuses can prevent an adversary from simply rewriting  $pk$  at a later time. Finally, each untrusted IC is paired with one (and one only) trusted IC.

A natural consequence of having a dual-IC system is that the ICs will require some form of integration. Whether the system is wirebonded together or makes use of vertical integration (e.g., through silicon vias), the authentication protocol remains the same. There are, naturally, integration choices that limit the performance of the Split-Chip interface, but this discussion is beyond the scope of this paper. Once the ICs have been integrated, an authentication protocol is executed on every power-on event. The authentication is performed in the following way:

1. Trusted IC reads the  $pk$  from the untrusted IC.
2. The received  $pk$  is validated against the stored  $pk$ .
3. If validation passes, it proceeds to the handshaking routine (see below). If it does not pass, then a locking mechanism is activated.<sup>2</sup>
4. If handshaking is successful, then the ICs are authenticated and the system can enter into the operational phase. If handshaking fails, then the same locking mechanism is activated.
5. For a pair of authenticated ICs, a secure channel can be initiated so application data can be exchanged. A symmetric encryption scheme such as the AES [21] is better suited—it is much lighter than asymmetric encryption and lends itself well for hardware implementation.

The following algorithm is proposed for handshaking. It relies on the generation of a random challenge that must match given the mathematical duality of the  $pk$  and  $sk$  keys. We refer to the trusted and untrusted ICs as T-IC and U-IC for conciseness.

<sup>2</sup>We do not implement any locking mechanism in this paper. Putting the IC into a simple passive or always-reset state can be sufficient, but it is likely that application-specific locks can be devised. Self destruction could be utilized, but the research on this topic is still on its infancy.

**Algorithm 1** Handshaking.

```

1:  $C \leftarrow rnd$   $\triangleright$  T-IC generates random challenge
2:  $C_{enc} \leftarrow RSA(C)$   $\triangleright$  T-IC encrypts challenge with  $pk$ 
3:  $U-IC_{input} \leftarrow C_{enc}$   $\triangleright$  T-IC sends encrypted challenge to U-IC
4:  $C_{dec} \leftarrow RSA(C_{enc})$   $\triangleright$  U-IC decrypts encrypted challenge with  $sk$ 
5:  $T-IC_{input} \leftarrow C_{dec}$   $\triangleright$  U-IC sends decrypted challenge back to T-IC
6: if  $C_{dec} = C$  then  $\triangleright$  If decrypted challenge matches the original
7:    $AUTH = 1$   $\triangleright$  Authenticated
8:   procedure  $secureChannel(pk)$   $\triangleright$  Initiate secure channel
9: else
10:    $LOCK = 1$   $\triangleright$  Enter locked state
11: end if

```

We mentioned that a shared key is necessary for a symmetric encryption channel to be established. The asymmetric scheme already in place can be used to exchange a shared key. This key should also be randomized for avoiding a bypass of the authentication protocol. The hardware module placed on the trusted IC that is responsible for generating a random challenge can be reused for this purpose.

**3.4.1 Implementation details - Untrusted IC**

Having the RSA key pair generated dynamically in the untrusted IC makes it challenging for an adversary to obtain or modify the key at runtime. This is a contrasting characteristic to a software-based RSA implementation. Furthermore, the secret key is regenerated every time the IC is powered on. To this end, we have designed a complete hardware RSA system including the key pair generation mechanism. Figure 5 shows the block diagram of this system.

As mentioned in Section 3.2, the RSA key pair is based on two prime numbers and the public exponent. The size of each prime number is half the size of the target key length (so that their product has a size that matches the target key length). In our approach, the prime numbers are generated in two steps. In the first step, we use a synchronous linear feedback shift register (LFSR) to generate pseudo-random numbers on demand. For our test implementation we assume 512-bit target key length and sampling width ( $s_w$ ) of 16 bits which require 256-bit primes and  $256 \times 16 = 4,096$ -bit PUF. As expected from a pseudo-random source, the LFSR will produce the same sequence of numbers for a given seed. In order to produce different sequences for different ICs, we use the SRAM PUF to

seed the LFSR. Then, in the second step, pseudo-random numbers are generated and tested to determine if a prime number has been found.

Even though simple primality testing algorithms are very well known (e.g., Sieve of Eratosthenes) and have a complexity of  $(O(n \cdot \log(\log(n))))$ , for the magnitude of the numbers involved in cryptography ( $\geq 2^{256}$ ), their runtime can be astronomical. One of the most efficient contemporary algorithms commonly used for primality test is the Miller-Rabin algorithm [22]. This is a probabilistic algorithm, meaning that it gives a probable result within probabilistic (but provable) bounds. More specifically, if a number does not pass the Miller-Rabin test, then it is provably a composite number. Otherwise, then the number is probably a prime.

The probability of Miller-Rabin to give an erroneous positive result, i.e., the number passes the test without being prime, has a worst case upper bound of 25%. Although this error rate appears to be quite high (even as an upper bound), in practice we can run the algorithm many times and thus effectively control the error rate at any desired target level. In this case the upper bound of error probability is formulated as  $Pe = 4^{-k}$  where  $k$  is the number of subsequent tries. In our design, we use a generous  $k$  value of 10, effectively setting the upper bound of error rate at  $2^{-20}$ .

Once the first prime number of length  $len$ -bits is generated, it is serially shifted to a shift register. The shift register can hold two prime numbers; therefore, its length is  $2 \times len$ . A flag is raised when the shift register is full, indicating that two prime numbers are ready for checking (not all prime numbers satisfy RSA requirements). The primality tester keeps requesting new pseudo-random numbers from the LFSR, then proceeds to check their primality and write them to the shift register until the full flag is raised.

The RSA validator unit takes both prime numbers as input. This unit performs a check to determine if the primes fulfill the requirements for RSA with respect to the chosen public exponent ( $e$ ). Given  $p, q$  as the prime numbers then

$$GCD((p - 1)(q - 1), e) \stackrel{!}{=} 1 \tag{1}$$

for valid RSA operation. The GCD can be found with the Euclidean algorithm. However, if  $e$  is also a prime number and smaller than  $(p - 1)(q - 1)$ , then the validity check can be further optimized to a much simpler integer division with remainder, since the GCD of any integer and a smaller prime is 1 if and only if the prime number does not divide the integer evenly. The most commonly used numbers for the public exponent are the first five Fermat numbers, 3, 5, 17, 257, and 65,537, due to the fact that they are primes and their binary representation contains mostly 0s and few 1s. This characteristic is helpful for the encryption



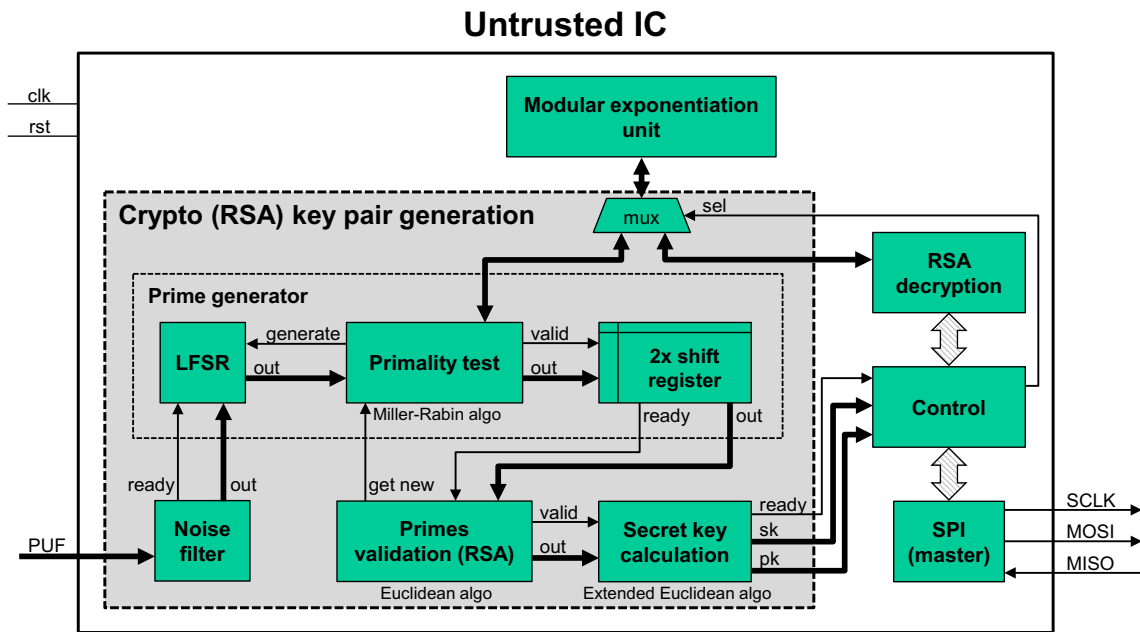


Fig. 5 Block diagram of the untrusted IC

process that follows (modular exponentiation). When the above equality does not hold, either a different prime pair or a different  $e$  can be tried. In our implementation,  $e$  is fixed in order to make the encryption process as efficient as possible. As stated in Algorithm 1, it is the trusted IC that must generate a challenge and encrypt it. Our goal is to make this encryption as lightweight as possible since the trusted IC of a Split-Chip system can and likely will be fabbed in a mature technology.

If the generated pair of primes does not pass RSA validation, then a new prime is requested from the LFSR, and the validation process starts from scratch. The new prime will be shifted in in order to replace one of the previously stored prime numbers. The new pair will then propagate to the RSA validator again. The process continues to iterate until a valid prime pair is obtained. In our results, we provide figures for the average number of tries of this iterative process.

Following the validation, keys are ready to be calculated. The “Secret key calculation” module in Fig. 5 calculates  $sk$  out of the generated prime pair. The modulus for  $pk$  has been already calculated in the validator unit since it is required for the validation ( $\phi(n) = pq - p - q + 1$ ). The validator actually outputs the prime numbers together with the modulus. The secret key is calculated as the modular multiplicative inverse of  $e \pmod{\phi(n)}$  using the extended Euclidean algorithm, i.e.,  $sk \equiv e^{-1} \pmod{\phi(n)}$ . When the calculation is done, the key pair generation unit raises the *ready* flag and outputs both keys which are captured by the control unit. The system is ready to start the authentication sequence.

Authentication is handled by the control unit which communicates with the trusted IC through an SPI interface. The intensive task of decrypting the received challenge is handled by an additional unit labelled “RSA decryption,” which implements large number modular exponentiation operation—a fundamental operation within an RSA cryptosystem. This module has been obtained from OpenCores [23], and is an open-source implementation of a full FIPS certified crypto-core for 512-bit RSA. The core implements 512-bit modular exponentiation using the Montgomery reduction method. This is the only significant portion of our system that was not in-house developed. However, we did modify the original IP as it was originally developed for FPGAs and relied on memory and FIFO instances that are FPGA-dependent.

Ideally, we would have implemented our authentication solution based on 1024-bit RSA since 512-bit RSA is currently considered subpar/outdated and can be factored with relative ease. Unfortunately, the core obtained from [23] is the only free IP for large number modular exponentiation we have access to<sup>3</sup>. Since the focus of this work is not on the public key cryptography algorithm per se but rather on the authentication sequence and hardware integration, we regard it as a necessary but fair choice. In any case, adopting 512-bit RSA allows us to compare our results with other research works and commercial products which offer the same encryption, as later shown in Section 4.

<sup>3</sup>Modular exponentiation is not offered as a DesignWare/ChipWare component by commercial synthesis tools, i.e., a third party IP is required.

The original implementation of [23] requires a pre-calculated, hard-coded Montgomery constant (based on the assumption that the keys are also known in advance — something that does not apply to our system). The Montgomery constant is given by

$$r_c = r^2 \text{ mod } m \tag{2}$$

where  $r = 2^{\text{wordLength}(\text{word\_count}+1)}$  and  $m$  is the modulus. Conversely, our custom ASIC implementation performs the Montgomery constant calculation on-the-fly. The modular exponentiation unit is the largest module, taking up ~52.7% of the total area of the untrusted IC. Since modular exponentiation is also required in the Miller-Rabin primality test and since the two modules (decryption and primality test) never run at the same time, we multiplex the unit between those to achieve a better design point in terms of area.

Communication between the two ICs is enabled through a custom SPI interface, achieving full duplex synchronous serial communication using only 3 wires. In contrast to typical SPI implementation, the slave (trusted IC) can send any amount of data to the master (untrusted IC) at any time—i.e., no predefined slave-to-master time windows or extra wires are needed. After authentication, the same channel can be utilized for any subsequent data exchange between the ICs. The SPI interface can be modified or replaced according to the latency and bandwidth requirements of the Split-Chip system.

### 3.4.2 Implementation Details - Trusted IC

The tasks of the trusted IC are summarized as (a) read  $pk$  from the untrusted IC and compare it with the stored  $pk$ , (b) generate a random challenge, (c) encrypt challenge with  $pk$ , (d) send encrypted challenge to the untrusted IC and wait for response, and (e) compare the received response with the original challenge. The circuitry in the trusted IC should be kept as minimal as possible due to the fact that it will be fabricated in an older, more mature technology that does not display the same high-performance achieved in the untrusted IC. For this reason, we use a small public exponent number  $e$  as previously mentioned (e.g., 3). Figure 6 depicts the block diagram of the trusted IC.

Upon power-up, the trusted IC waits for the untrusted IC to finish its own power-up sequence and send the generated  $pk$ . After  $pk$  is received, the authentication sequence is started, as described in Section 3.4. Due to the deterministic nature of the untrusted IC power-up sequence (and arrival of  $pk$ ), time cannot be used as a seed for generation of a random challenge. While an LFSR is capable of generating a random challenge, it would potentially make the authentication scheme weaker if challenges were also deterministic. Instead, a true random number generator

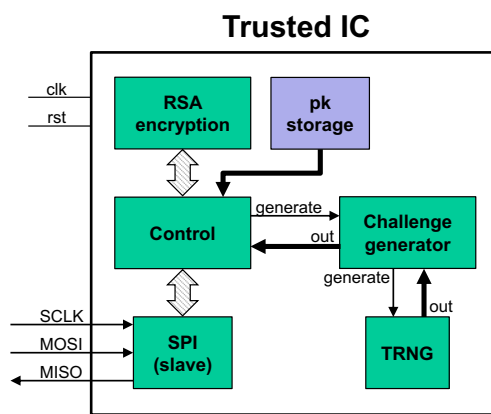


Fig. 6 Block diagram of the trusted IC

device can be used in lieu of an LFSR. A magnetic device like the one proposed in [24] can be utilized with minimal impact on area or performance.

Verilog source code for the trusted IC, untrusted IC, and accompanying communication interface can be found in our public repository [25]. We now proceed by discussing the specifics of the hardware implementation and presenting our results.

## 4 Implementation Results and Discussion

The untrusted IC has been designed in a commercial 16nm FinFET process. For the trusted IC, the technology of choice is a planar 28-nm CMOS process. This combination of processes was selected based on the availability of commercial IP that we have access to. Synthesis was performed using standard cell libraries and compiler-generated memories specific to each process. Relying on commercial IP (instead of academic or predictive technologies) allows us to present numbers that are representative of the chosen nodes.

### 4.1 PUF Silicon Measurements

As the PUF is at the center of our design and drives several of its choices, we herein present measurement data obtained from actual silicon. In [26], we taped out several designs using the same commercial FinFET technology we adopted for the untrusted IC of this paper. Such designs make extensive use of SRAM memories, which we embedded with read-out circuitry to allow them to specifically work as a PUF and to expose the PUF contents to the outputs of the chip. The memory elements are 8T SRAM cells, therefore cross-coupled inverters are the source of its entropy. The 8T cells are packed together by a memory compiler obtained from a third party. The chosen PUF has

no special features/modification for increased stability or security.

For establishing the quality of the PUF, we first performed a series of measurements by power cycling the tested chips and capturing their signatures. Each chip has been power cycled 10 times, which we considered sufficient to discern good PUF bits from random bits. During each power cycle, 144 bits are read from the memory and serialized to appear at a primary output of the chip. These are recorded and compared across power cycles. A total of 12 chips were tested, accounting for 17,280 bits captured. On average, 11.4 bits per chip (7.6%) present random behavior, as detailed in Fig. 7. All tested dies come from the same lot/wafer, therefore the differences can be attributed to die-to-die process variation.

Furthermore, we also performed sanity checks on the obtained signatures. Bits that present a random behavior were not discarded during this process. The most basic characteristic we evaluated is the ratio of 1s and 0s, for which we detected a large bias towards 0s. Of the 17,280 collected bits, 5,131 bits are 1s, a ratio of approximately 29.6%. For one of the tested chips, the number of 1s is only 376 for all 10 reads, a ratio of only 26.1%. The same chips were also tested at voltages as low as 0.45 V, well beyond the acceptable margins for a nominal voltage of 0.85 V. No significant influence coming from voltage shift was detected. Finally, in Fig. 8 we provide a heatmap for the occurrence of 1s and 0s. Each region represents a single bit that was read 10 times from each of the 12 chips, for a total of 120 reads per bit. Since the PUF is strongly biased, several bits have a very low rate of 1s and appear as almost white in the image. We highlight that our proposed filter can easily manage this behavior by tweaking the threshold  $q_{th}$ .

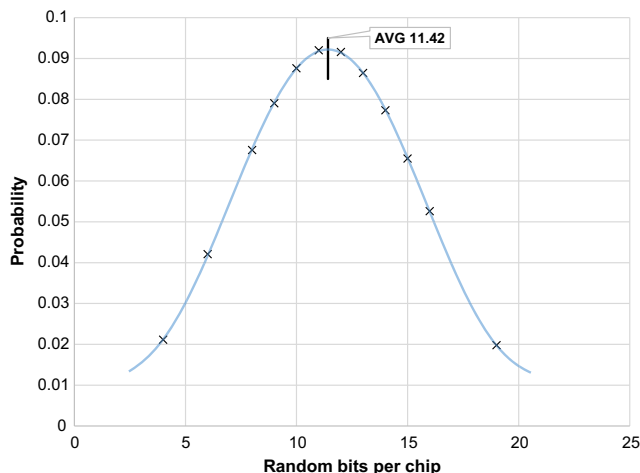


Fig. 7 Distribution of the number of random bits per chip. The blue trendline reveals an almost ideal normal distribution

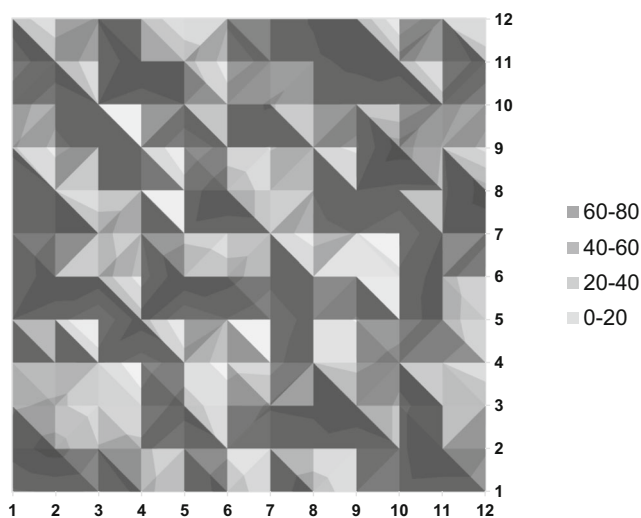


Fig. 8 12 × 12 matrix representation of the 144 bits collected from multiple chips. The darker the region is, more often 1s were collected from that particular memory element

### 4.2 Power, Performance, and Area

Our designs for both ICs were synthesized separately but using the same tool and optimization options. Figures for power, performance, and area (PPA) are collected and shown in Table 2. We synthesize our designs for three clock speeds: (a) at maximum operating frequency, (b) at 500 MHz, and (c) at 100 MHz. The key length is set at 512 bits. The maximum frequency is 1.5 GHz for the untrusted IC and 1.36 GHz for the trusted IC.

It must be noted that the authentication framework, even when amply optimized, requires a considerable amount of hardware resources. For maximum frequency, the combined cell count is greater than 200 K gates. Not all applications can tolerate this overhead for authentication. However, modern SoCs are large by nature and would be able to

Table 2 Power, performance, and area results for the complete system placed into the untrusted and trusted ICs

		Untrusted IC (16 nm)	Trusted IC (28 nm)
Power (mW)	@max freq.	122.6	119.4
	@500 MHz	47.4	58.3
	@100 MHz	17.4	15.3
Std. cell count	For max freq.	120,539	88,279
	For 500 MHz	95,603	72,729
	For 100 MHz	92,245	70,544
Total area (μm <sup>2</sup> )	For max freq.	87,987	153,834
	For 500 MHz	77,892	125,340
	For 100 MHz	76,884	123,512

tolerate this overhead, especially if security is a concern (e.g., dependable and military systems).

### 4.3 Runtime Characteristics

Due to the non-deterministic nature of the prime number generation, a careful assessment of the design characteristics is necessary. We test various key lengths up to 512 bits. We run 1,000 simulations per key length from which we extract statistical runtime figures. We account for the randomness of the SRAM PUF by using a random seed for each simulation run. For all simulation runs performed, no instances of erroneous RSA key generation or faulty authentication result were registered.

The time required to read the raw PUF data is proportional to the SRAM word size and the number of bits necessary to build a post-quantization key of the desired length. In our implementation, we consider a 4,096-bit SRAM is necessary for a key length of 512. Assuming a word size of 32 bits, 128 clock cycles are required to read all addresses of the SRAM. The filtering process (key reconstruction) is completed by the time needed to read  $\{key\ length / 2\}$  bits from the OTP eNVM plus 2 clock cycles. The OTP eNVM can be read at the same time as the SRAM, incurring no additional clock cycles for reading it. Runtime results for finding a prime and also for validating the pair of primes are shown in Table 3. These are, by far, the most time consuming tasks of the entire authentication scheme and are executed only on the untrusted IC.

Aside from the key pair generation, the remaining operations executed by the untrusted IC, (i.e., send  $pk \rightarrow$  receive challenge  $\rightarrow$  decrypt challenge  $\rightarrow$  send response) require exactly 197,514 clock cycles for a 512-bit key. The trusted IC, on the other hand, can execute its operations (i.e., receive  $pk \rightarrow$  validate  $\rightarrow$  generate challenge  $\rightarrow$  encrypt  $\rightarrow$  send  $\rightarrow$  validate answer) in 2,955 clock cycles for a 512-bit key. All these operations are completely deterministic and thus have a fixed clock count, unlike the key pair generation which is stochastic.

Further tuning of the circuitry is possible by choosing not only a key length, but also a target time-to-authenticate. For example, if a 512-bit key length is selected and one second is allowed for authentication, the clock frequency

can be as low as 120 MHz when accounting for one standard deviation of the key pair generation time. We highlight that there is a one-time cost to authentication at power-up, and that no system performance degradation is accrued from having longer or shorter authentication times.

### 4.4 Comparison with Software-Based Solutions

As an alternative to the hardware only implementation described in this paper, a software-based implementation of RSA can be sought with the use of a processor. Naturally, the key benefit of a software solution is the flexibility to change the running code. Not only can the authentication code be modified as necessary, but the processor can be reused to run any other algorithm that is specific to a given system.

There are a number of drawbacks, however, which makes this option unattractive for certain applications. When security is concerned, a hardware-based implementation is considered more resilient. See [27] for a thorough discussion on attacks to software systems. Generally speaking, to mount an attack on hardware is a difficult task as altering or corrupting silicon is no trivial task. On the other hand, it is reasonably easier to alter algorithms that run on general-purpose processors. Moreover, software-based solutions are, by and large, inefficient compared with hardware implementations. This translates to a larger PPA product for performing the same task.

To assess the PPA of a software-based solution, we synthesize an ARM-compatible 32-bit processor obtained from [28]. Since an instance would be required for each IC, we synthesize the core in both target technologies. To approximate the memory requirements for both stack and heap, we wrote and compiled a C++ program that generates random prime numbers, checks their validity, and subsequently runs RSA encryption and decryption. The program was specifically compiled for ARM’s instruction set using the compiler available from [29]. We selected the public exponent  $e = 3$  to match the hardware implementation.

We assume the authentication/communication between ICs would incur a small overhead of 10% to the code size. In summary, the software performs nearly all the tasks of

**Table 3** Runtime results for finding primes and validating they are adequate for RSA

RSA key length	32	64	128	256	512
Average tries to find a prime	4.6	10.2	20.9	46.3	95.5
Average tries for valid pair	5.9	5.6	5.8	5.9	6.2
Average number of clock cycles ( $\times 10^6$ )	0.64	1.51	4.54	16.48	66.27
Std. dev. of number of clock cycles ( $\times 10^6$ )	0.53	1.22	4.00	15.06	51.10

**Table 4** Power, performance, and area results for the Amber core implementation

		Untrusted IC (16nm)	Trusted IC (28nm)
Power (mW)	@max freq.	80.2	53.5
	@500 MHz	38.3	27.2
	@100 MHz	9.1	8.2
Std. cell count	For max freq.	23,001	25,685
	For 500 MHz	19,366	21,511
	For 100 MHz	19,096	21,077
Total area ( $\mu\text{m}^2$ )	For max freq.	83,750	160,236
	For 500 MHz	82,370	153,851
	For 100 MHz	82,273	153,092

the hardware-based authentication method. The memory technology is assumed to be embedded SRAM. Memory modules have been generated for each IC using specific compilers for each technology. Our results are summarized in Table 4. The maximum frequency varies by a small amount, as the untrusted IC can achieve 1,085 MHz and the trusted IC can achieve 1,020 MHz.

By comparing the results of Tables 2 and 4, it can be seen that the hardware implementation outperforms the processor/software solution in area and maximum operating frequency. The software solution consumes less power, however, with a power difference in the range of 19–53%.

However, we cannot assess the performance difference without runtime figures for the software solution (i.e., clock or instruction count). The authors of [30] report that an optimized software implementation of RSA for a 512-bit key, when executed on an ARM Cortex-A9 (a much larger and powerful processor than Amber), requires between  $4.63 \times 10^6$  and  $3.7 \times 10^6$  clock cycles.<sup>4</sup> This clock cycle count refers to the RSA decryption part alone. Our hardware implementation requires 197,514 clock cycles for the equivalent RSA decryption (more than 23 times faster), which would certainly offset the increase in power.

In Table 5, we list the memory requirements for the code we have written, which are not negligible. This memory would most likely be SRAM. However, the code itself would also have to be stored, for which some form of ROM could be used. The data previously shown in Table 4 does not account for either memory. Our estimation is that the results from Table 4 would have to be significantly inflated, clearly making our specialized hardware solution stand out.

<sup>4</sup>The clock cycle count varies depending on the use of external hardware other than the processor itself. The reported algorithm is based on Montgomery multiplication with Orup's algorithm in RELIC [31].

**Table 5** Memory characteristics of our C++ code for various key lengths

Key length	Max heap size (bytes)	Stack + heap size (bytes)
256 bits	2,648	7,432
512 bits	2,696	8,488
1,024 bits	2,792	11,152
2,048 bits	3,392	26,320

#### 4.5 Comparison with Other Research Works

Execution time of the authentication scheme is dominated by the key pair generation process, which in turn is dominated by the execution time of the primality test. Our primality test process takes 99,096 clock cycles for 256-bit numbers and 198,016 for 512-bit numbers. We run the same test up to 10 times for generating a reasonably reliable prime (Miller-Rabin test), albeit this number can be lowered depending on the target confidence level and target time-to-authenticate. In general, the authentication runtime is limited by the choice and implementation of primality test algorithm.

In order to situate our work with respect to other implementations, we now provide a discussion on the few works that provide implementation details of hardware-based RSA solutions. Lu C. et al. in [32] report 3.96-s runtime for generating a 512-bit prime with optimal settings, requiring 25.3 calls on average to their primality test. Their hardware runs at 3.57 MHz which renders  $\sim 558,700$  clock cycles to complete each test; more than double when compared to our implementation. Their total performance though, with only  $\sim 14 \times 10^6$  clock cycles for a 512-bit prime, is much better than in our case due to the use of a sieve function—a variation of sieve of Eratosthenes. A sieve function can detect and exclude a portion of composites that contain small factors very fast. It requires the storage of a set of small prime numbers which uses for trial divisions. Implementing a sieve function increases the overall performance by reducing the average number of calls to the actual primality test. The authors in [32] use a set of 2,560 primes in their sieve function, effectively eliminating 85.7% of composites. The drawback of the sieve function is that it requires extra memory for storing the set of primes, as well as the intermediate values. In our case, we opted not to use a sieve function as it does not allow us to remove/reduce any other modules from our system. A sieve function only accelerates the prime validation process at the expense of area.

Cheung et al. in [33] present an optimized prime number validation that exploits parallelism and extra memory to

reduce the overall number of clock cycles. They report 256-bit prime number validation within a range of  $0.07 \times 10^6$  to  $3.55 \times 10^6$  clock cycles and 512-bit numbers in the range of  $0.27 \times 10^6$  to  $23.64 \times 10^6$ . Their approach introduces a significant area (and consequently power) overhead.

Joye et al. in [34] report 3.15 seconds in average to generate a 512-bit prime on a 3.57 MHz crypto-processor (roughly  $11.2 \times 10^6$  clock cycles). Their method relies on some precomputed elements which they store in an additional 2.7-KB memory. They use the weak Fermat pseudoprime test with 1 base, as opposed to the strong Fermat pseudoprime test (i.e., Miller-Rabin) with 10 bases that we use.

Bahadori et al. in [35] report an average of 2.85 seconds for 512-bit key pair generation with 10% of the workload assigned to a 4 MHz processor and 90% assigned to a 32-MHz crypto-coprocessor. This would amount to an equivalent of 29-MHz processing power and about  $82.65 \times 10^6$  clock cycles. The authors use a sieve function with 5,120 primes. Our implementation performs the same task in  $\sim 66.27 \times 10^6$  clock cycles without a sieve function.

Also worth comparing is Infineon's SLE-66CX160S chipset, a popular high-end security cryptocontroller used in a myriad of devices. The IC employs an 1,100-bit Advanced Crypto Engine capable of performing RSA encryption/decryption for 160 – 1,024-bit keys. In spite of currently being outdated and replaced with newer models for larger key lengths (starting from 1,024 bits), there is still some merit in comparing its performance with our 512-bit RSA cryptosystem. The IC completes 512-bit RSA decryption in 550,000 clock cycles, more than 350,000 additional clock cycles compared with our implementation.

## 5 Conclusion and Future Work

In this paper, we demonstrated an active authentication scheme between ICs fabricated in a Split-Chip fashion. As untrusted fabrication remains a reality for most governments and private entities, concerns like overproduction and IP theft arise. Split-Chip naturally thwarts IP theft, while the scheme presented in this paper prevents the adversary from overproducing the untrusted IC as it will fail to authenticate with existing trusted ICs. Furthermore, the proposed scheme does not rely on software or processing cores, as the entire hardware is custom designed for the specific task of authentication. A comparison with other works in the area revealed that our achieved performance is often many times superior.

There are a number of improvements that can be devised for the herein presented scheme. For instance, a lightweight version of the authentication would prove useful for smaller systems, as well as support for longer keylengths for higher

security guaranties. A promising direction for the latter is the conversion of the system to utilize Elliptic Curve Cryptography instead of RSA. Another improvement would be the design of a different PUF filter which would not rely on volatile digit mapping and would not have to trade entropy for reproducibility, or do so more efficiently. We intend to investigate these possibilities as future avenues of research.

**Funding Information** This work was supported in part by the Defense Advanced Research Projects Agency under contracts FA8750-17-1-0059 “Obfuscated Manufacturing for GPS (OMG)” and HR001116C0038 “Circuit Realization at Faster Timescales (CRAFT)”.

**Open Access** This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

## References

1. Crawford M, Telesco T, Nelson C, Bolton J, Bagin K, Botwin B (2010) Defense industrial base assessment: counterfeit electronics. In: Reports of the U.S. Department of Commerce
2. Skorobogatov S, Woods C (2012) Breakthrough silicon scanning discovers backdoor in military chip. In: Cryptographic Hardware and Embedded Systems – CHES 2012
3. Levin C et al (2012) Inquiry into counterfeit electronic parts in the Department of Defense supply chain. In: Senate Report of the Committee on Armed Services
4. Tehranipoor M, Koushanfar F (2010) A survey of hardware trojan taxonomy and detection. *IEEE Des Test Comput* 27(1):10–25
5. Chakraborty RS, Bhunia S (2009) HARPOON: An obfuscation-based SoC design methodology for hardware protection. *IEEE Trans Comput-Aided Des Integr Circ Syst* 28(10):1493–1502
6. Roy JA, Koushanfar F, Markov IL (Oct 2010) Ending piracy of integrated circuits. *Computer* 43(10):30–38
7. Vaidyanathan K, Das BP, Sumbul E, Liu R, Pileggi L (2014) Building trusted ICs using split fabrication. In: IEEE Int. Symp. on Hardware-Oriented Security and Trust (HOST)
8. Pagliarini S, Mai K, Blanton S, Mitra S, Pileggi L (2018) Split-chip design for hardware security. *IEEE Design and Test* (under consideration)
9. Sweeney J, Zackriya M, Pagliarini S, Pileggi L (2019) Securing digital systems via Split-Chip Obfuscation. In: GOMACTech Technical Program
10. Herder C, Yu M, Koushanfar F, Devadas S (2014) Physical unclonable functions and applications: a tutorial. *Proc IEEE* 102(8):1126–1141
11. Devadas S, Suh E, Paral S, Sowell R, Ziola T, Khandelwal V (2008) Design and implementation of PUF-based “Unclonable” RFID ICs for anti-counterfeiting and security applications. In: 2008 IEEE International Conference on RFID
12. Layman PA, Chaudhry S, Norman JG, Thomson RJ Electronic fingerprinting of semiconductor integrated circuits, Sep 2002. U.S. Patent 6738294
13. Su Y, Holleman J, Otis B (2007) A 1.6pJ/bit 96% Stable chip-ID generating circuit using process variations. In: IEEE International Solid-State Circuits Conference, pp 406–611

14. Holcomb DE, Bursleson WP, Fu K (2007) Initial SRAM state as a fingerprint and source of true random numbers for RFID Tags. In: Conf. RFID Security, pp 11–13, Malaga, Spain
15. Maes R et al (2012) Experimental evaluation of physically unclonable functions in 65 nm CMOS. In: 2012 Proceedings of the ESSCIRC, pp 486–489
16. Handschuh H, Schrijen G, Tuyls P (2010) Hardware intrinsic security from physically unclonable functions. In: Towards Hardware-Intrinsic Security: Foundations and Practice, pp 39–53
17. Diffie W, Hellman ME (1976) Multiuser cryptographic techniques. In: National Computer Conference and Exposition. ACM, New York, pp 109–112
18. Rivest RL, Shamir A, Adleman L (1978) A method for obtaining digital signatures and public-key cryptosystems. *Commun ACM* 21(2):120–126
19. Guajardo J, Kumar SS, Schrijen G, Tuyls P (2007) FPGA Intrinsic PUFs and their use for IP protection. In: Cryptographic Hardware and Embedded Systems - CHES 2007, pp 63–80
20. Holcomb DE, Bursleson WP, Fu K (2009) Power-Up SRAM state as an identifying fingerprint and source of true random numbers. *IEEE Trans Comput* 58(9):1198–1210
21. Specification for the Advanced Encryption Standard (AES) (2001) Federal Information Processing Standards Publication 197
22. Rabin MO (1980) Probabilistic algorithm for testing primality. *J Number* 12(1):128–138
23. Villar JC 512 bit RSA algorithm. [https://opencores.org/project/rsa\\_512](https://opencores.org/project/rsa_512). Online; accessed August 2018
24. Bhuijn S et al (2017) A self-calibrating sense amplifier for a true random number generator using hybrid finFET-straintronic MTJ. In *IEEE/ACM, Int. Symp. on Nanoscale Architectures (NANOARCH)*, pp 147–152
25. Karageorgos I (2019) Split-chip authentication. [https://github.com/ecelab-org/Split-Chip\\_authentication](https://github.com/ecelab-org/Split-Chip_authentication)
26. Pagliarini SN, Isgenc MM, Martins MGA, Pileggi L (2018) Application and Product-Volume-Specific customization of BEOL metal pitch. *IEEE Trans Very Large Scale Integr (VLSI) Syst* 26(9):1627–1636
27. Yuze B, Schaumont P, Witteman M (2018) Fault attacks on secure embedded software threats, design, and evaluation. *J Hardw Syst Secur* 2(2):111–130
28. Conor Santifort Amber core. <https://opencores.org/project/amber>. Online; accessed August 2018
29. ARM Limited. GNU Embedded Toolchain For ARM <https://developer.arm.com/open-source/gnu-toolchain/gnu-rm>. Online; accessed August 2018
30. Sharif MU, Shahid R, Gaj K, Rogawski M (2016) Hardware-software codesign of RSA for optimal performance vs. flexibility trade-off In *Int. Conf. on Field Programmable Logic and Applications (FPL)*, pp 1–4
31. Aranha DF, Gouvêa CPL RELIC toolkit. <https://github.com/relic-toolkit/relic>
32. Lu C, dos Santos ALM, Pimentel FR (2002) Implementation of fast RSA key generation on smart cards in ACM symp. on applied computing, SAC '02, pp 214–220
33. Cheung RCC, Brown A, Luk W, Cheung PYK (Dec 2004) A scalable hardware architecture for prime number validation In *IEEE Int. Conf. on Field-Programmable Technology*, pp 177–184
34. Joye M, Paillier P, Vaudenay S (2000) Efficient generation of prime numbers. In: *Cryptographic Hardware and Embedded Systems — CHES 2000*, pp 340–354
35. Bahadori M et al (2010) A novel approach for secure and fast generation of RSA public and private keys on SmartCard. In: *IEEE Int. NEWCAS Conference*, pp 265–268

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.