Key Recovery for Content Protection Using Ternary PUFs
(A Comparative analysis between SRAM and Pre-Formed ReRAM)

Abstract: Physical unclonable functions, embedded in terminal devices, can be used as part of the recovery process of session keys that protect digital files. Such an approach is only valuable when the physical element offers sufficient tamper resistance. The ternary cryptographic protocols presented in this paper, leverage the physical properties of resistive random-access memories operating at extremely low power in the pre-forming range. The most unstable cells during key generation cycles are masked to reach bit error rates (BERs) in the 10^{-3} range. We propose replacing the error correcting codes with light search engines, that use ciphertexts as helper data to reduce information leakage. The tamper-resistant schemes discussed in the paper include: (i) a cell-pairing differential method to hide the physical parameters; (ii) an attack detection system and a low power self-destruct mode; (iii) a multi-factor authentication, information control, and a one-time read-only function. In the experimental section, we describe how prototypes were fabricated to test and quantify the performance of the suggested methods.

1. Introduction

Physical elements, physical unclonable functions (PUFs), tags, and other hardware systems have been proposed to secure both terminal devices and their digital files because they can be tamper resistant. Recent patents are proposing ways to protect information [1–4], this list is not exhaustive, given the importance of the topic. This study is targeting networks of devices, the "internet of things" (IoT), interacting with service providers. We propose end-to-end solutions that are based on the recovery of cryptographic keys from tamper-resistant PUFs. The mainstream error correcting codes (ECC) are replaced by a search engine, the response-based cryptography (RBC). We detail how the ternary PUFs can be implemented with pre-formed resistive random-access memories (ReRAM).

1.1. One Way Unclonable Functions

The one-way unclonable functions "**Ψ**" in this paper, are defined as functions generating a stream of bits "**K**" from a random number "**T**", and individual digital access instructions "***IDAccess***"; **K** ← **Ψ (T, *IDAccess*)**. The function **Ψ** is kept secret in such a way that the knowledge of **T** and ***IDAccess*** does not disclose **K**; therefore, it is assumed that **(T, *IDAccess*)** can become public information. The knowledge of **K** never discloses the input parameters (**T**, ***IDAccess***). The function is unclonable, making it highly unlikely to be duplicated. During "enrollment", the image of the one-way function of the client device is downloaded in a look-up table of the server. This allows the server to communicate safely with the client device as both parties can independently generate the same stream **K** from the shared input parameters **T** and ***IDAccess***. The stream **K** is part of a cryptographic protocol:

- **T** is XORed with password **PW** ;The result is hashed (SHA-3) to generate the message digest **MD**, and an extended output function (SHAKE) pointing at a set of addresses **A**; **A** ← XOF ← **MD** ← Hash (**T** ⊕ **PW**).
- ***IDAccess*** is used to retrieve the set of instructions "**I**" needed to generate **K** from the set of addresses **A**. To enhance security, ***IDAccess*** is XORed with **MD**; $\mathbf{I} \leftarrow (\textit{IDAccess} \oplus \mathrm{MD})$. "**I**" are masking the fuzzy cells to reduces the BERs of **K**.

The one-way unclonable functions can be implemented with PUFs, exploiting nanocomponents that are unique and unclonable due to small variations occurring in their fabrication. PUFs are described by the one-way function of f converting n-bit challenges **C** = {c_1; …; c_i; …; c_n} in m-bit responses **K**= {k_1; …; k_j; …; k_m}; c_i and $k_j \in \{0, 1\}$.

C = (**T**, ***IDAccess***) is the stream of challenges, while **K** is the stream of responses. During key generation cycles, the responses of **K** should match the ones generated during enrollment; **K** = **Ψ** (**C**) ← **C**.

The protocols based on PUFs are effective with error-correcting schemes to handle aging, and environmental effects [5–8]. Examples of implementation with various PUFs are summarized in Table 1.

Ring oscillator PUFs. Ring oscillator (RO) PUFs are designed typically with CMOS-based circuits, each oscillate at a slightly different value due to small variations during fabrication [9]. The pairing of two rings generates a consistent response, 0 or 1, if the first ring oscillates slower or faster than the second one. RO-based PUFs are widely used to secure field programable gate array circuits (FPGA). They can be subject to side-channel analysis through electro-magnetic interference. The set of addresses **A** generated from **T** can point to a set of RO pairs. The set of instructions **I** point to a subset of ROs, and to a targeted value of the power supply. The responses of the PUF are the stream **K**.

Arbiter PUFs. Arbiter PUFs are designed with chains of multiplexer (MUX) circuits, each allowing the transmission of electronic signals through two possible paths, up or down [10]. The chains feed Reset-Set latches, which switch to 0 or 1 when the delay at the Reset pad is either faster or slower than the delay at the Set. The stream of addresses **A**

generated from **T**, point to a set of instructions that drive the MUXs. The set of instructions **I** point to a subset of instructions in order to generate the responses **K**.

SRAM-based PUFs. Each cell of a static random-access memory (SRAM) is a flip-flop that has an equal chance to wake as a 0 or 1 after a power-off—power-on cycle [11–13]. Most of the cells tend to wake in the same way, thereby creating a fingerprint of the device. These PUFs are widely used, most electronic components already contain arrays of SRAM cells. The set of addresses **A** generated from **T** can point to a set of cells, the set of instructions **I** point to a subset of the array to generate the responses **K**. The entropy is strong when the SRAM array is large; however, methods to read the memory are available to the opponents.

ReRAM-based PUFs. This PUF has interesting tamper-resistant features [14,15]. Each cell of a resistive random-access memory (ReRAM) has a unique resistance value that is compared to a median value. The value, 0 or 1, is generated from each cell depending on whether the resistance has been lower or higher than the median. The set of addresses **A** generated from **T** point to a set of cells, the instructions **I** point to a subset of the array to generate the responses **K**. Other devices such as DRAMs [16], Flash [17–19], and MRAMs [20,21] are used to design PUFs.

Table 1. Examples of challenge-response configurations for various PUFs.

PUF	Challenges			Responses
	T	***IDAccess***		
RO	To point at a set of M pairs of ROs	To point at a subset of N pairs of RO (N < M)	To avoid pairs oscillating at a similar frequency	Each N pair of ROs generates a 0 or 1
Arbiter	M sets of instructions driving MUXs in the up or down position	To point at a subset of N instructions (N < M)	To avoid the sets of instructions known to be unstable	Each N set of instructions generates a 0 or 1
SRAM	To point at M addresses in the SRAM array	To point at a subset of N addresses (N < M)	To avoid the SRAM cells known to be unstable	Each N cell of SRAM array generates a 0 or 1
ReRAM	To point at M addresses in the ReRAM array	To point at a subset of N addresses (N < M)	To avoid the ReRAM cells known to be unstable	Each N cell of ReRAM array generates a 0 or 1

The addition of a third state allows the tracking of the portions of the PUF that should be masked because they are fuzzy, marginal, unstable, or fragile. The objective is then to reduce BERs and improve reliability [22,23]. A thorough enrollment cycle to identify these "weak" portions and their masking during response generation, results in a better quality PUF, requiring little to no error-correcting scheme for cryptographic key generation. The knowledge of the addresses with fragile physical elements that can be damaged during normal operations have tamper-resistance properties [15]. A cryptographic protocol generating keys solely from the addresses of PUFs that are known to be reliable does not damage the weak portions of the PUF. An opponent trying to interact with the PUF without such knowledge could damage the PUF, leaving behind traces of the attack. The addition of an additional state has been successfully integrated into PUF-based key generation schemes [23]. The keys generated by the addressable PUFs are varying with when the currents injected change [15].

1.2. Error correcting Schemes

Error-correcting schemes mitigate the differences between responses generated on-demand, and those collected during enrollment [24–28]. The responses of a PUF must be perfectly corrected to be used as cryptographic keys.

Error-correcting codes (ECC). As shown in Fig. 1, the challenges transmitted by the server (**T**, ***IDAccess***) allow the generation of the response **K'** from the PUF, which should be similar to the initial responses **K** extracted from the information stored in a secure database by the server during enrollment. ECC is needed to uncover the stream **K** from the potentially erratic stream **K'**, and to generate error-free cryptographic keys. ECC exploits data streams called "helpers" to correct the erratic bits. In certain applications, the data helpers need to be protected by encryption schemes to avoid leakages to the opponents [29]. A fuzzy extractor embedded in the client device reads **K'** and the data helper to find **K**. The computing power consumed by the fuzzy extractor could be prohibitive for power-constrained IoTs, making these devices vulnerable to side-channel analysis.

Response-based cryptography (RBC). RBC enhances the protection of the key generation process by eliminating the need to operate a fuzzy extractor at the client device level, see Fig. 2. In lieu of generating a data helper from **K**, the client device hashes **K'** to generate the message digest **H(K')**, which is transmitted to the server. The RBC is a search engine that is able to recover **K'** from **K** and the message digest **H(K')** [30–32]. Through an iterative process, the RBC search engine hashes the initial response **K** = $\mathbf{K_0}$ and compares the message digest $\mathbf{H(K_0)}$ and **H(K')**. If these two do not match, the RBC tests all data streams $\mathbf{K_{1k}}$ from a Hamming distance of 1 of $\mathbf{K_0}$. If these do not match, the RBC iterates and tests all data streams $\mathbf{K_{2k}}$ from a Hamming distance of 2. Such an iterative process can handle Hamming

distances up to 3, and 256-bit long responses. Beyond this, the latencies become prohibitive. A scheme fragmenting the data streams with nonces has been documented as effective with BERs as high as 20%. For example, if 256-bit long responses are impacted by 7% BERs, a fragmentation of 4 is applied; with SHA-128, **H(K')** as a 4 × 128-bit long stream. Each fragment is filled with nonces and is hashed. The RBC can benefit from high-performance computing (HPC), graphic processor units (GPU), and associative processor units (APU) with parallel computations [33,34].

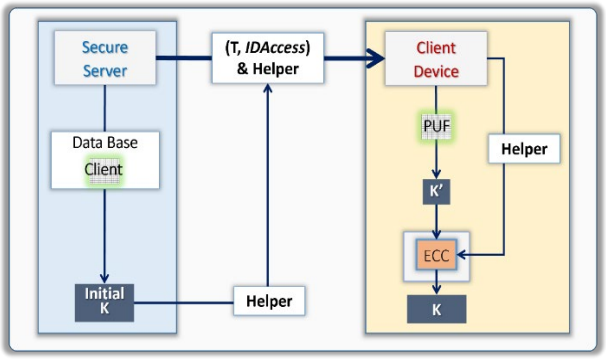

Figure 1. Block diagram of an ECC scheme.

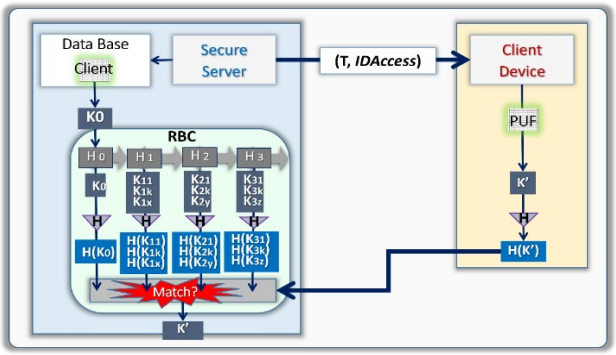

Figure 2. Block diagram of RBC search engine scheme.

2. Session key recovery with ternary PUFs

Preparation cycle- Session key encapsulation. The protocol shown in Fig. 3 encrypt a session key **Sk** with the PUF. The information stored in the database is considered public information. Without the PUF, or its image, the secret key **Sk** should not be retrievable. During the preparation phase, the challenges (**T**, ***IDAccess***) generating the responses **K** from the database are transmitted to the client device by the secure server. The data stream ***IDAccess*** allows the masking of the fuzzy positions, which has the objective of reducing BERs at the client level. This requires an enrollment cycle to identify the erratic cells. The session key **Sk** is then encrypted by the client device using the freshly generated responses **K'**. The information stored by the client device is (**T**, ***IDAccess***) the challenges, **H(K')**, the hash message digest of the response **K'**, and **E(Sk, K')** is the ciphertext generated by encrypting the **Sk** with **K'**.

Session key recovery. In the key recovery cycle, see Fig. 4, the challenges (**T**, ***IDAccess***) are retrieved by the client device, and used to generate responses **K"** from the PUFs that are not always identical to **K'** due to the drifts of the PUF. The search engine uses the message digest H(**K'**) to retrieve **K'** from **K"**. Other ECC schemes can replace the search engine in the architecture presented in Fig. 4. The session key **Sk** is recovered by decrypting the ciphertext with **K'**. The session key can be used in various schemes such as AES and DES, or in public key infrastructures such as Elliptic curves, RSA, Dilithium, Kyber, NTRU, Falcon, Saber, Rainbow, and Classic McEliece [35–42].

Light search engine implementation. A light version of the RBC was developed for the ReRAMs because the BERs are low enough. The search was limited to Hamming distances of 0 and 1. When no message digest matches **H(K')**, rather than exploring higher Hamming distances, the process iterates through a new query of the PUF with the same challenges (**T**, ***IDAccess***) to generate new responses.

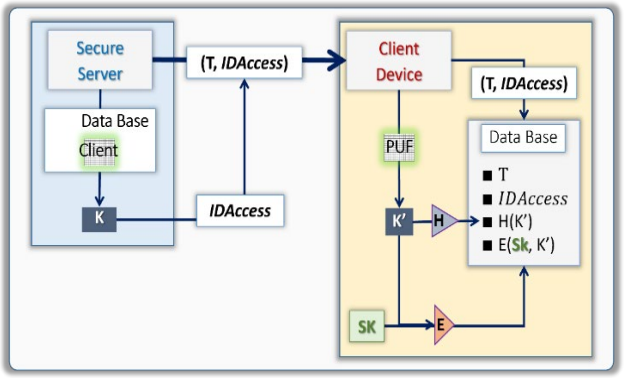

Figure 3. Encapsulation of the session key **Sk**.

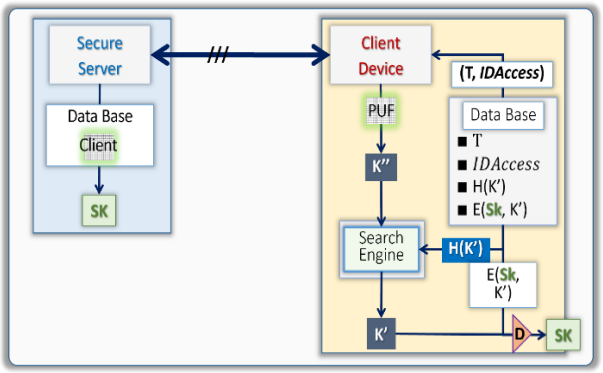

Figure 4. Recovery of the key **Sk**.

3. Content protection with ternary PUFs

One way to protect digital files is to encrypt them with a cryptographic key, such as the session key **Sk** presented in Section 3. The same session key can be used to either encrypt or decrypt multiple digital files. In this section, we present methods for delivering and protecting digital files, each of which is protected by its own set of keys. Rather than using on-demand challenge-response pairs (CRPs) to protect session keys, we suggest using a different CRP for each digital file. The random number **T** and the individual digital access parameter ***IDAccess*** of each CRP generates the response **K** protecting a particular digital file **M**. As shown in Figure 5, the process of preparing the delivery of a digital file **M** starts with the generation of a set of challenges (**T**, ***IDAccess***). **T** is obtained through a random number generator, and ***IDAccess*** is computed from the look-up table containing an image of the PUF. This allows for the masking of the ternary positions, as well as the parameters such as the electric current that is injected during the key generation. The digital file **M** is encrypted with responses **K**. An example of the protocol that encrypts and transmits the ciphertext to a client device is as follows:

- Both communicating parties have independent access to a shared password **PW**; number **T** and **PW** are XORed. The resulting stream is hashed with a SHA-3 generating **MD**, which is extended with a SHAKE to generate stream **A** for the m addresses; **A** ← SHAKE(**MD**) ← **MD** ← SHA-3 (**T** ⊕ **PW**)
- The m-bit long mask is retrieved from ***IDAccess*** to hide the addresses containing fuzzy positions. This leaves k positions, k < m, for response generation from the image of the PUF. The output is the k-long response **K**.
- The digital file **M** is encrypted into ciphertext **C** with **K**, the responses **K** are hashed with a SHA-3 to get **H(K)**, and the mask is XORed with **MD**. **T**, **C**, and **H(K)** are transmitted to the client device.

The suggested protocol assumes that **T**, **C**, and **H(K)** can be transmitted through non-secure channels. The information is protected by the password, ***IDAccess***, the PUF, or its image to disclose the digital file **M**. To trigger the read cycle, the server communicates the missing piece of the challenges, ***IDAccess***, which the client device combines with **T**, see Fig. 6. The one-way unclonable function generates responses **K'** from the PUF, providing the information to the client device to generate the key **K** from **K'** ← **Ψ (T, *IDAccess*)**. The search engine recovers the initial responses **K** from **K'** and the message digest **H(K)**. This allows access to the digital file **M** by decrypting the ciphertext **C = E(M,K)**.

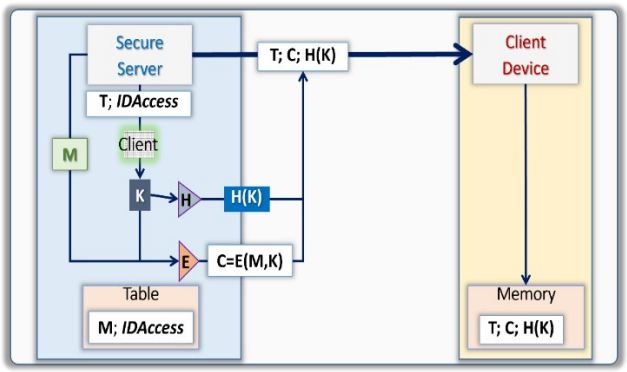

Figure 5. Encryption of digital file M.

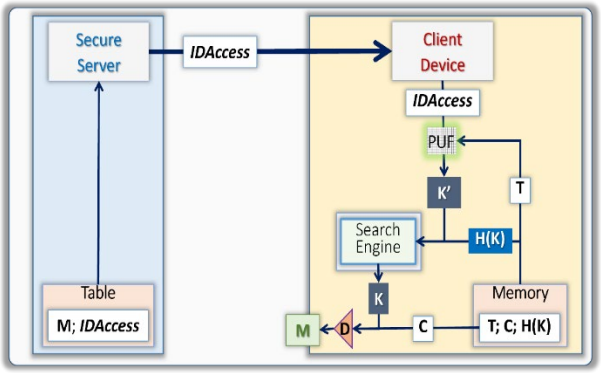

Figure 6. Decryption of **M**.

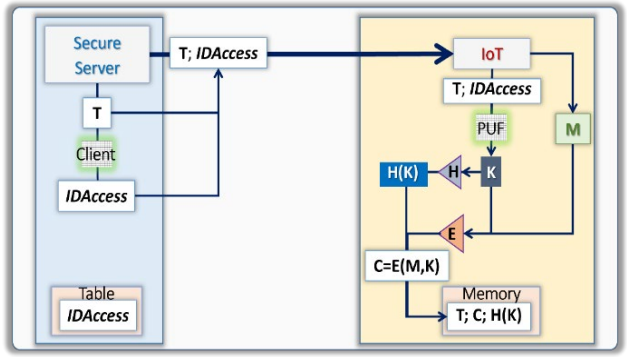

Figure 7. Encryption of file M for IoTs.

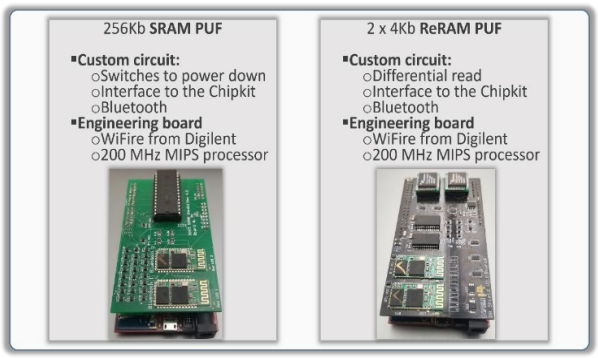

Figure 8. HW set up for the SRAM (left), and the ReRAM (right).

Protection of digital files stored by IoT terminals. The goal is to prevent third party from using the information stored in the memory unit of the IoT to access the session key. During the preparation cycle, as shown in Fig. 7, the server generates the challenges (**T**, ***IDAccess***) from the image of the PUF. The client device operates as follow:

- Retrieves the challenges (**T**, ***IDAccess***) to generate the responses **K**.
- The file **M** is encrypted with **K** to generate the ciphertext **C**. The IoT hashes **K** for the search engine.
- The IoT stores **T**, **C**, and the message digest **H(K)**, but not ***IDAccess***, which is stored by the server.

The IoT can only decrypt message **M** after receiving ***IDAccess*** from the server. If the one-way function is tamper-resistant, an opponent cannot decrypt **M** without an image of the one-way unclonable function. An example of the implementation of the decryption of **M** by the device is as follows:

- Receive ***IDAccess***; read from the memory number **T**, ciphertext **C**, and message digest **H(K)**.
- Generation of **K'** from the one-way unclonable function; retrieve **K** from **K'** and **H(K)** with a search engine.
- Decrypt the digital file using **K** as a cryptographic key.

4. Implementation with static RAM devices

The implementation with SRAM is straightforward, as the devices are commercially available, while the implementation with pre-formed ReRAM requires the design of custom circuits. Both are interfacing with the Wi-FIRE ChipKit engineering board from Digilent. The core engine of the engineering board is a microcontroller manufactured by Microchip with a 200 MHz 32-bit MIPS processor, a 2 MB embedded Flash, and a 500 KB SRAM. The board is powered via a USB port, and it operates at 3.3 volts. As shown in Fig. 8, the custom boards with memory devices are plugged into the ChipKit board. A set of switches allows quick power-off cycling of the SRAM to reset the device prior to the response generation. The read cycles of the SRAM are fast, 256 bits needs 10 µs. Most SRAM cells always wake in the same state, as a "0" or a "1" after power off—power on cycles; however, 3 to 5 % of the array changes states in each cycle. The experimental results are shown in Fig. 9. The results represent 30 successive key generation cycles, lasting 2 s each; the enrollment had 100 cycles at room temperature. At each key generation cycle, the server randomly selects 512 positions, of which 256 are masked to keep only the most stable cells. The keys generated from the look-up table are on the left, those from the PUF are on the right, the count of errors is last. The average BERs were about one percent. Such BERs are well within the capability of the RBC, with false reject rates (FRR) below 1%, and one-second average latencies. After about 100 cycles, we were left with approximately 88 % of the cells waking in the same state, the rest of the population had at least one erratic response. At between 100 cycles and 1000 cycles, less than 5% of the additional cell population was still unstable. The results presented below in Fig. 10 were derived by varying the number of enrollment cycles at different temperatures. We performed tens of thousands of key generation cycles to obtain statistically significant results. The SRAM was subjected to 1000 enrollment cycles: 0 °C in light blue, 20 °C in red, 40 °C in grey, 60 °C in orange, and 80 °C in dark blue. The BER was then computed with responses generated in the 0 °C to 80 °C range for each temperature of enrollment.

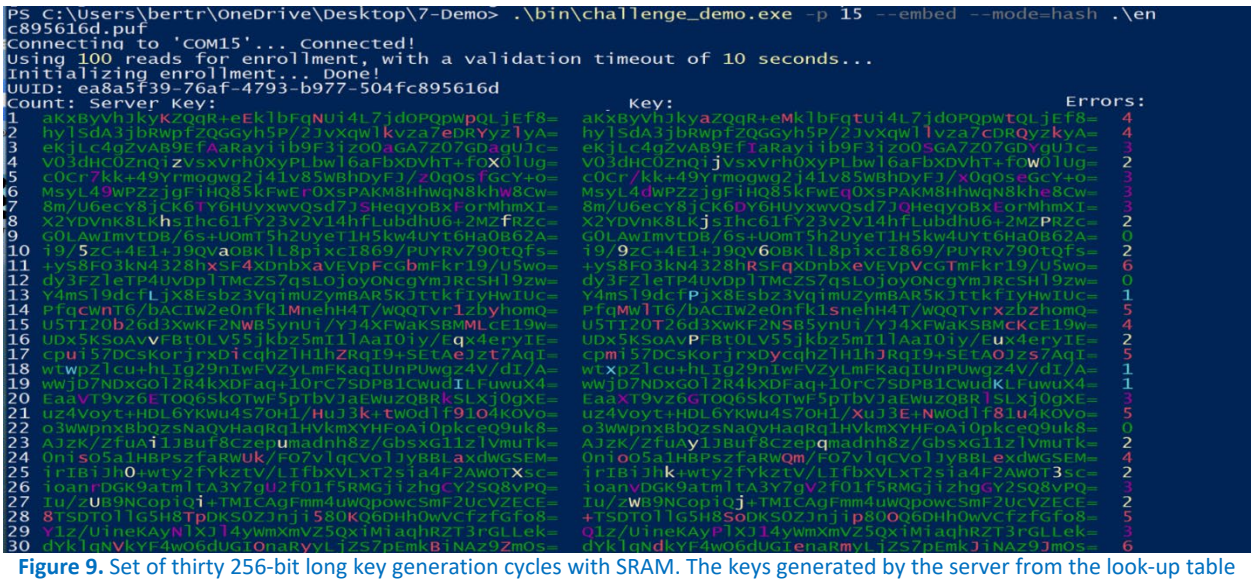

Figure 9. Set of thirty 256-bit long key generation cycles with SRAM. The keys generated by the server from the look-up table are on the left, those generated from the SRAM PUF are on the right, followed by the count of errors.

We observed smaller BERs, in the 2 10^{-5} range, when the responses were generated at the same temperature as the enrollment. A multi-temperature enrollment, shown in green in Fig. 10, yielded BERs in the 1 10^{-5} to 2 10^{-6} range. The multi-temperature enrollment took 5 × 5000 s, or 7 h.

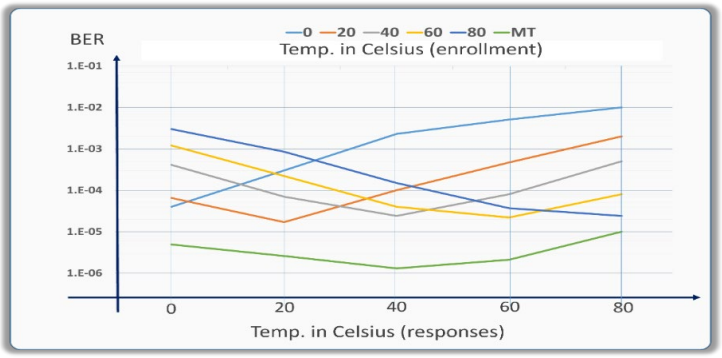

Figure 10. BER of the SRAM PUF (Y-axis). The enrollment consisted of 1000 power off/on cycles at 0 °C, 20 °C, 40 °C, 60 °C, 80 °C, and at multiple temperature. The responses were generated at 0 °C, 20 °C, 40 °C, 60 °C, and 80^{0}C (X-axis).

5. Implementation with resistive RAM devices

The design with two 4 Kbit ReRAM circuits allowed faster read cycles of about 10 ms for a 256-bit long key. The responses were generated by injecting a small current from 100 nA and 800 nA, in pairs of cells, each located in separate arrays of 4 k cells. The number of challenge-response pairs in this experiment was 8 × 4096 × 4096 = 128 million. The response was a "0" when the first cell had a resistance lower than the second cell, and a "1", in the opposite configuration. The results shown in Fig. 11 represent 30 successive key generation cycles, lasting 2 s each. The starting point for the protocol is that 512 pairs of cells are randomly selected, of which 256 pairs act as a buffer, keeping the remaining 256 pairs further apart in resistance values. The 256-bit long keys generated from the look-up table are on the left, those from the ReRAM-PUF are on the right, followed by the count of errors, which are low. In Fig. 12, an experiment to quantify the effect of the size of the buffer is presented. To generate 256-bit long keys, 256 + **k** pairs were selected, with **k** being the number of extra pairs, with the lowest differences in resistance values, that were removed. The goal was to remove enough pairs to generate 256-bit keys with BERs in the part per million (ppm) range. The lowest buffer sizes, around 29 pairs, were observed at room temperature and 800 nA. The largest buffer sizes, around 35 pairs, were observed at 80 °C and 100 nA. The protocols presented in the experimental section of this paper used 256 pairs as a buffer, which was anticipated to generate BERs way below 1 ppm.

```
PS C:\Users\bertr\OneDrive\Desktop\7-Demo> .\bin\challenge_demo.exe -p 13 --embed --protocol=cellpairing
dbf-f105-41e8-a4d4-402e17aedfb5\die1.puf .\enroll\7257cdbf-f105-41e8-a4d4-402e17aedfb5\die2.puf
Connecting to 'COM13'... Connected!
Using 80 reads for enrollment, with a validation timeout of 10 seconds...
Initializing enrollment... Done!
UUID: 7257cdbf-f105-41e8-a4d4-402e17aedfb5
Count: Server Key:                                                    Key:                                                    Errors: Current
1  vrK+/Pan+/5WcO53J/hf2HIlY+x3hb5mG9XnvCufxXE=  vrK+/Pan+/5WcO53J/hf2HIlY+x3hb5mG9XnvCufxXE=  0  688
2  X/3+265X/v+u5pbr83f/NU386//Z2/+7rvifu7H9sfg=  X/3+265X/v+u5pbr83f/NU386//Z2/+7rvifu7H9sfg=  0  608
3  4cHVLz1Br/PqxduTVqp61wf8uo//T+77/ry89CU//z8=  4cHVLz1Br/PqxduTVqp61wf8uo//T+77/ry89CU//z8=  0  503
4  BINw4AAQUClwmUYWgCqLt6xgI5AJACASASBRCQG5hIA=  BINw4AAQUClwmUYWgCqLt6xgI5AJACASASBRCQG5hIA=  0  608
5  EbQyEJ5hzBds1pdNHrQdeF2ZRqsopVgBQLKB3jZqSt4=  EbQyEJ5hzBds1pdNHrQdeF2ZRqsopVgBQLKB3jZqSt4=  0  794
6  vWHTXBMVk43pqpd87fDMHlUHZcHWZP2a9UdHKtl/gDs=  vWHTXBMVk43pqpd87fDMHlUHZCHWZP2a9UdHKtl/gDs=  0  794
7  O/z/I37fW//Ddt4/wP3wvv8/7e9++zt9/dzuTj5xffY=  O/z/I37fW//Ddt4/wP3wvv8/7e9++zt9/dzuTj5xffY=  0  304
8  EpwDawBCYgA1EASGcUxwQoYESIsGICiREiEKAKrA4Cw=  EpwDawBCYgA1EASGcUxwQoYESIsGICiREiEKAKrA4Cw=  0  503
9  qU3QJYBrB5jRfgFJVlIgkrOSAYJQSfgpbaTVG6JAoBg=  qU3QJYBrB5jRfgFJVlIgkrOSAYJQSfgpbaTVG6JAoBg=  0  688
10 P+PAr8abPm1/EN4O/b9e8h7tR27TYzm/PXzmffnMKd8=  P+PAr8abPm1/EN4O/b9e8h7tR27TYzm/PXzmffnMKd8=  0  688
11 dO7WUQoPkiSjdRO1FfmPUGgG7DtcG6jYTfH+5sznWFO=  dO7WUQoPkiSjdRO1FfmPUGgG7DtcG6jYTfH+5sznWFO=  0  794
12 DSMIGXYAAkACMJh1OiCgFmgQEQE3AsqlSoSCBIMBwEA=  DSMIGXYAAkACMJh1OiCgFmgQEQE3AsqlSoSCBIMBwEA=  0  304
13 qAAAAAAEEAAQACAAAGAAAAACQUAgAACIAMCAkAABEBA=  qAAAAAAEEAAQACAAAGAAAAACQUAgAACIAMCAkAABEBA=  0  106
14 AMAAoABIFgEMAAAIEgIBACEQAIAAAAgAAAABAAgBEAA=  AMAAoABIFgEMAAAIEgIBACEQAIAAAAgAAAABAAgBEAA=  0  106
15 6/v/5/77rarP5n/d+/22+ze//u/Ub7rrf6fL/+5v5ec=  6/v/5/77rarP5n/d+/22+ze//u/Ub7rrf6fL/+5v5ec=  0  304
16 NEDAGFSCxYBaICTKTRQhRZByhDOGCGIhwAGBETIJkCU=  NEDAGFSCxYBaICTKTRQhRZByhDOGCGIhwAGBETIJkCU=  0  503
17 cuOCPWz/MTtVA4OWvmDwtGua37XyTCfWvyK7aEsNPPk=  cuOCPWz/MTtVA4OWvmDwtGua37XyTCfWvyK7aEsNPPk=  0  794
18 zjnmhXnrF3qLqR53PcxeH/ObfW1trXrukvLdFp7okn8=  zjnmhXnrF3qLqR53PcxeH/ObfW1trXrukvLdFp7okn8=  0  397
19 sHGAQAAAQJEAxCQQgwCzggrqwGBBIYCQQgUAooARAAA=  sHGAQAAAQJEAxCQQgwCzggrqwGBBIYCQQgUAooARAAA=  0  304
20 YAAC0CSKJBQQ4GJAkChBGYQAaIV5SETRAQFIMVSZhCo=  YAAC0CSKJBQQ4GJAkChBGYQAaIV5SETRAQFIMVSZhCo=  0  608
21 AAABCADAAAAASIgACIQEAAAgEAgQgIAAAACAACAEBQE=  AAABCADAAAAASIgACIQEAAAgEAgQgIAAAACAACAEBQE=  0  106
22 P5s7ldnzdpcYPIV1ofdtbDwg3FJSmQrsIXvO7yxVODE=  P5s7ldnzdpcYPIV1ofdtbDwg3FJSmQrsIXvO7yxVODE=  0  794
23 /1oVrX4OOwyzvs/t5pxu/77/u/u+fM1F293/ed38//8=  /1oVrX4OOwyzvs/t5pxu/77/u/u+fM1F293/ed38//8=  0  304
24 CAAhgAAhAIAAKAUBQQDIAAAAAAASAAAAkCEAGIAIAAA=  CAAhgAAhAIAAKAUBQQDIAAAAAAASAAAAkCEAGIAIAAA=  0  106
25 QEmAUWJgaC7AAOKhgnAQY4gOzUBgXNAIJihh5qKCMiY=  QEmAUWJgaC7AAOKhgnAQY4gOzUBgXNAIJihh5qKCMiY=  0  503
26 AXXD+HCX/m+e7/Sqqa8qW9mpzb55z868v/O5K9X3f38=  AXXD+HCX/m+f7/Sqqa8qW9mpzb55z868v/O5K9X3f38=  1  688
27 OU6XHP6+cGWHtHc/C298TfeiX/Ol7O54L5wnKeUfNj8=  OU6XHP6+cGWHtHc/C298TfeiX/Ol7O54L5wnKeUfNj8=  0  688
28 9u3G8bf/uOP/fr73n77yg/+x/L9L3r3wn++3/2+9//4=  9u3G8bf/uOP/fr73n77yg/+x/L9L3r3wn++3/2+9//4=  0  304
29 e258rc/J/94Pcj/5/4r/fX9z3+N/vX893eXJ+8/fZvI=  e258rc/J/94Pcj/5/4r/fX9z3+N/vX893eXJ+8/fZvI=  0  304
30 +N85m/3/+v+f819v5vf1vz5P9cq3/73s923vY5mfnfO=  +N85m/3/+v+f819v5vf1vz5P9cq3/73s923vY5mfnfO=  0  304
```

Figure 11. Set of thirty, 256-bit long key generation cycles, with two ReRAM devices. The keys generated by the server from the look-up table are on the left, those generated from the ReRAM PUF are on the right, followed by the count of errors.

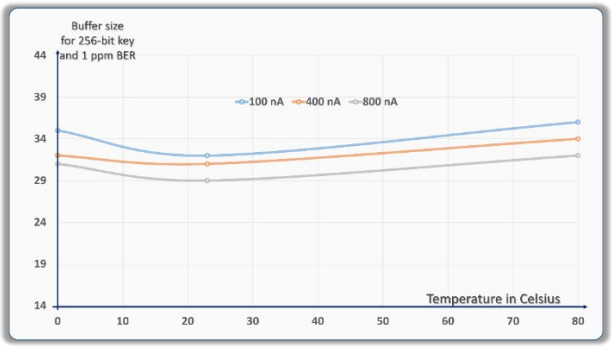

Figure 12. Size of the buffer (Y-axis) to get a BER at 1 ppm.

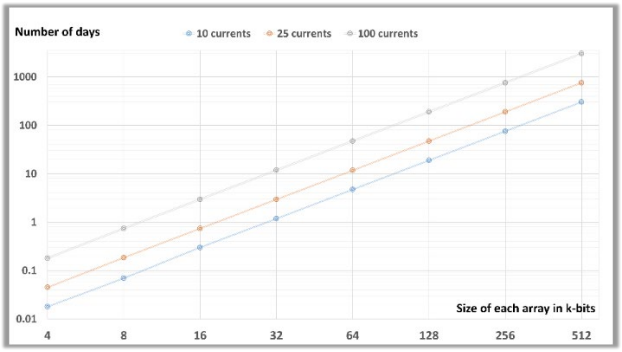

Figure 13. Days for the crypto-analysis of pairs of ReRAM.

The differential protocol we developed for the two ReRAM circuits also protects the system against an opponent trying to read the resistance values of the ReRAM arrays. After enrolment, the two circuits can be mounted in such a way that only differential measurements can be conducted. In Fig. 13, the number of days needed to read all pairs, as a function of the size of the array, is shown, together with the number of possible currents injected into each cell. It takes 1000 days to read all pairs produced by two arrays of 512 K-bits each with 25 possible levels of currents.

Comparing SRAMs and ReRAMs for key generation. SRAM PUFs are widely available, cheap, easy to use, and fast [43]. This is a perfect technology to use to implement the methods presented in this paper, namely, key recovery, content delivery, and digital file protection. One of the tradeoffs of using SRAM PUFs is the additional cost of enrollment that is needed to operate at low BER. The repetitive power off/on cycles are time-consuming. The ReRAM technology has higher potential to design tamper-resistant solutions, as it is summarized in Table 2.

Table 2. Comparison between SRAM and pre-formed ReRAM based key generation protocols.

Factor	256 Kb SRAM	2 × 4 Kb ReRAM
Commercial availability	Broad	Limited
Entropy: number of cells/pairs	256 k cells	128 M pairs
BER responses	$2\ 10^{-6}$	$1\ 10^{-8}$
Latencies: enrollment cycle	7 h	15 min
Latencies: responses/256 bits	10 µs	10 ms
Crypto-analysis	5 min	4.4 h
Sense attack	No	Yes
Self-destroy	No	With 1.5 V
Radiation hardness	Limited	Yes

- Entropy: number of cells or pairs: The differential protocol comparing the resistance between the cells belonging to 4 Kb ReRAM arrays involves 16 million pairs, 8 different currents, the number of CRP reach 128 million.
- Bit error rates of the responses: As shown in Figure 10, BERs in the $2\ 10^{-6}$ range is possible with SRAMs at a cost of enrollment cycles lasting multiple hours, which lacks practicality. The way to reduce the BERs of two ReRAM arrays, driven by the differential protocol, is to increase the size of the buffer. An extrapolation of the data reported in Fig. 12, points to BERs in the $1\ 10^{-8}$ range.
- Enrollment cycles: There is no need to test the pairs of ReRAM cells upfront during enrollment, testing each array thoroughly is enough to generate the initial response from a look-up table. The measurement of the cells is analog; there is no need to repeat the measurements to quantify the proportion of "0" or "1".
- Response cycles: Generating responses from the SRAM PUF is extremely fast after powering on the device. We also checked that 10 ms are enough to read 256-bit long streams with ReRAMs with a differential circuit.
- Crypto-analysis: The full SRAM array can be red in seconds after power off/on cycles. A crypto-analysis of the ReRAM requires the analysis of 128 million pairs, which takes about 4.4 h.
- Ability to sense attacks: The design of sensing elements inserted in the ReRAM arrays has been reported [15]. An opponent exploring the ReRAM arrays without knowledge of the vulnerable cell population will damaging these cells. It is possible to monitor the potential infiltration of a crypto-analyst and to detect an attack.
- Self-destruct mode at low power: In case of an attack, the user can trigger a self-destruct mode of the ReRAM cells by initiating the forming cycles.
- Radiation hardness: SRAMs are vulnerable to ionizing radiation. The ReRAM technology is rad-hard [44].

The SRAM technology for the design of outstanding PUF-based cryptosystems has been demonstrated by industrial suppliers and academic institutions.

6. BERs during key recovery with ReRAM devices

The ReRAM PUFs were subjected to 80 successive reads in order to identify the noisy reading cases. The key recovery protocols from a single IoT device are much more challenging, the BERs can be higher as the initial responses are the result of a single read, which could be noisy. We conducted 44 key recovery cycles to estimate the BERs, see Fig. 14, in which the PUFs are used twice; the left column displays the keys generated during the first read from the PUFs and the right column shows the keys generated from the same PUFs during the recovery cycles. The numbers of errors between the two are shown in the last column. The resulting BERs are well within the RBC-light capabilities. We performed 5000 successive cycles, the results are summarized in Fig. 15, which shows on a log scale the probability of the occurrence of erratic keys as a function of the numbers of errors for 256-bit long responses. The average BERs observed in this experiment were 6.148 10^{-4}. These BERs created 787 bit-errors out of the 5000 pairs of 256-bit long keys. The distribution of erratic keys observed here is described by a Poisson distribution with parameter λ equal to the average number of erratic bits per key:

$$\lambda = 256 \times BER = 256 \times 6.148\ 10^{-4} = 0.1574 \quad (5)$$

4290 keys (85.8%) have 0 errors (Poisson: 85.4%); 643 keys (12.9%) have 1 error (Poisson: 13.4%); 58 keys (1.2%) have 2 (Poisson: 1.06%); 8 keys (0.16%) have 3 errors (Poisson: 0.06%); 1 key (0.02%) have 4 errors (Poisson: 0.002%). The RBC-light that searches for erratic keys with a Hamming distance of no more than one is successful 98.5% of cases. The need to generate a second response occurs in only 1.5% of the cases, and a third response in 0.02% of cases. False reject rates (FRR) of the key recovery will occur if the latencies of the RBC are too long due to an excessive number of iterations, and lengthy PUF response generation cycles. Out of 5000 key recovery cycles, 4290 keys had zero errors, 643 keys had one error, 58 had two errors, eight had three errors, and one key had four errors.

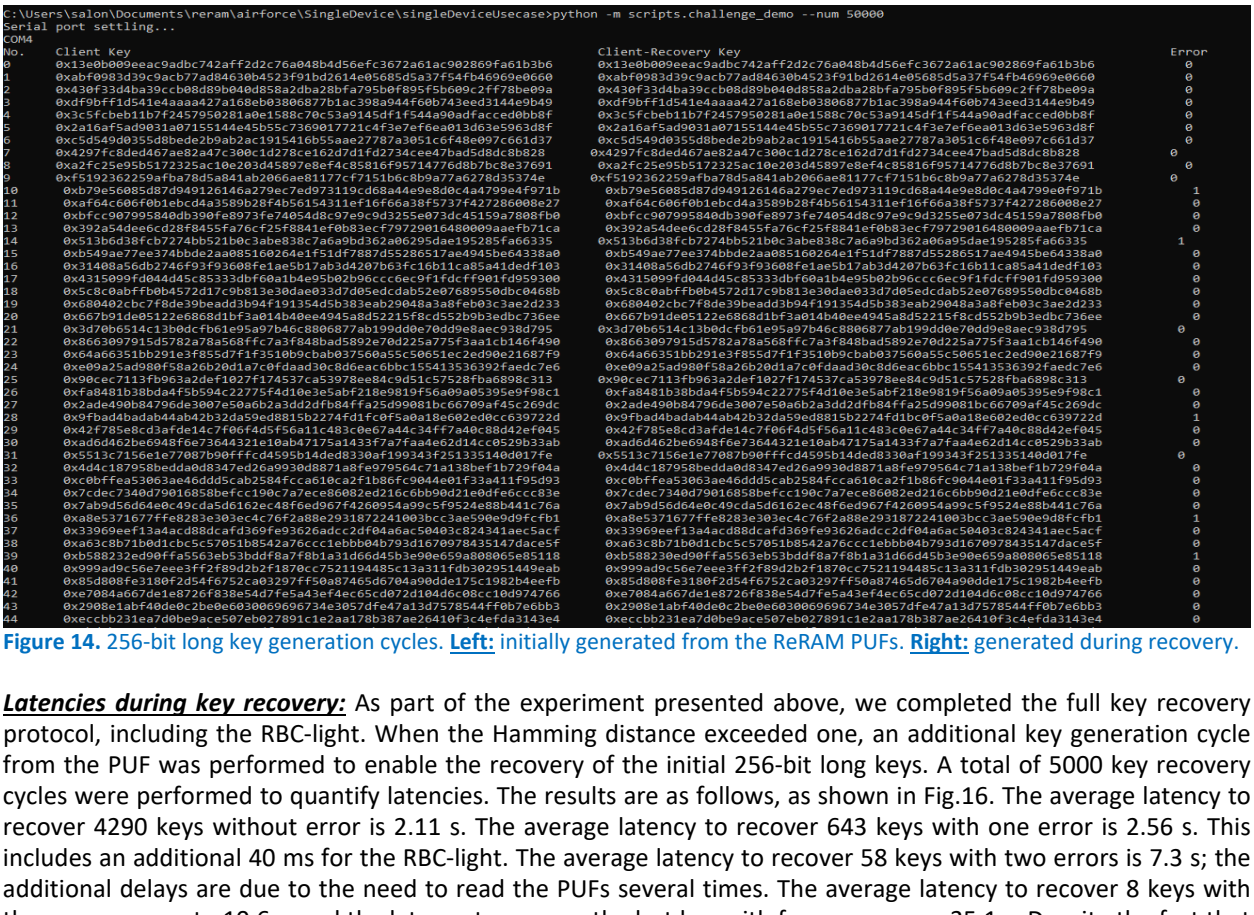

Figure 14. 256-bit long key generation cycles. **Left:** initially generated from the ReRAM PUFs. **Right:** generated during recovery.

Latencies during key recovery: As part of the experiment presented above, we completed the full key recovery protocol, including the RBC-light. When the Hamming distance exceeded one, an additional key generation cycle from the PUF was performed to enable the recovery of the initial 256-bit long keys. A total of 5000 key recovery cycles were performed to quantify latencies. The results are as follows, as shown in Fig.16. The average latency to recover 4290 keys without error is 2.11 s. The average latency to recover 643 keys with one error is 2.56 s. This includes an additional 40 ms for the RBC-light. The average latency to recover 58 keys with two errors is 7.3 s; the additional delays are due to the need to read the PUFs several times. The average latency to recover 8 keys with three errors rose to 10.6 s, and the latency to recover the last key with four errors was 35.1 s. Despite the fact that the hardware that we designed for this study is far from being optimized in terms of stability, noise, and latency, the overall performance of the key recovery schemes is in line with the study objectives. With this protocol, 100% of the 5000 keys were recovered, with an average latency of 2.25 s.

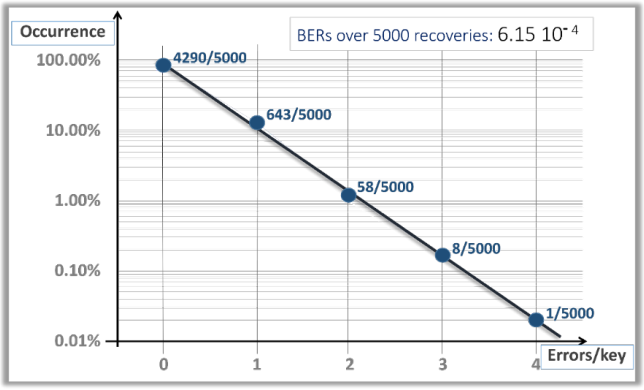

Figure 15. Percent of erratic keys generated with ReRAM PUFs.

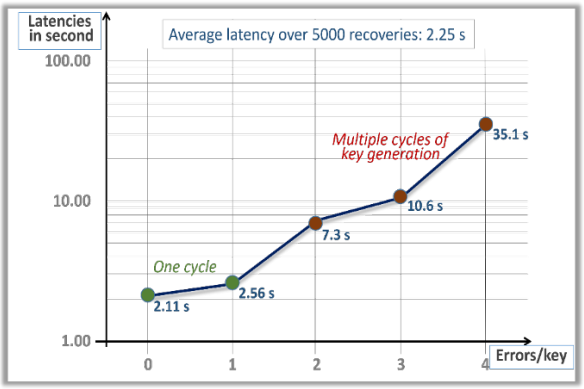

Figure 16. Latencies for the key recovery cycles.

Software considerations: The development of the protocols presented in this section was mainly based on generic software, that can eventually be implemented in hardware with secure components. The XOR functions concatenated the input parameters, such as random numbers, with passwords for multi-factor access control. The hash function SHA-3 (512 bit), with its one-wayness, is at the core of several layers of protection, including:

- To convert the XORed input parameters into the message digest MD, and SHAKE, to selects the addresses of the PUF used for response generation. MD is used to protect IDAccess after XORing operations.
- As part of the RBC, the hashing of the responses is used to uncover the original responses.

The XOF SHAKE converted the MD into the set of addresses pointing to the PUF. The session keys were encrypted using AES-256, and the keys were generated from the original responses of the PUF. They were decrypted with the keys retrieved from the RBC-light, and with the fresh responses from the same PUF after RBC correction. The session keys generated from the pre-formed ReRAM PUFs were tested successfully as seed data to generate private and public key pairs for elliptic curve cryptography, Dilithium [37], and Saber [39].

Summary and Future Research

The analysis reported in this paper show that the proposed ReRAM based solutions outperform SRAM PUF-based schemes in terms of BERs and tamper resistance. Bit error rates below the 10^{-3} range were demonstrated in the keys generated from ReRAMs operating in the pre-forming range with the differential cell pairing protocol. Unlike with SRAMs, such low BERs do not necessitate lengthy enrollments to remove the unstable cells. The differential protocol keeps only those pairs of cells with resistances further apart from each other. The BERs are reduced when the proportion of pairs that are masked by the protocol is large enough. Low BERs enable the use of a small search engine, the RBC-light, as a replacement for power-consuming error-correcting schemes, without fuzzy extraction, and without data helpers. The combination of tamper resistance, low BERs, and low power correcting methods facilitated the development of an end-to-end cryptographic system to deliver and protect digital files. We intend to perform exhaustive characterizations of BERs of the ReRAMs, with additional enrollment cycles, extended temperature cycles in the −40 °C to +140 °C range, increased buffer sizes of the number of pairs, and accelerated aging cycles. The methods presented in this paper could be considered for the following applications:

- Per paid content delivery. A service provider can deliver several encrypted files containing information such as movies, music, apps, maps, and operating systems. The user obtains access to the files after paying a fee.
- Protected user manuals. Staged access to a prepared set of instructions for a particular task, which evolves over time, due to changes in conditions. The users receive, as needed, access codes to open a particular portion of a user manual. An example of such an application would be pilots flying a plane.
- Cooperative users. The server concurrently sends to user 2 the information needed by user 1 to retrieve a sub-key, and vice-versa. The full key is generated by knowledge of both sub-keys.
- Securing interconnected IoTs. Nodes of IoTs such as controlling and metering elements in a grid, home hubs, smart sensors, contain information that is stored locally, and which needs to be protected constantly.
- Authentication of the server. When operating in a zero-trust environment, the server sends users information previously used to encrypt and store a session key.

References

1. Wu, P.; Nathan, R.; Tredennick, H. Secure Hardware Signature and Related Methods and Applications. US Patent 10,891,366, Jan. 12, 2021.
2. Kameo, N.; Anzai, F.; Nishimae, E. Information Distribution Device, Distribution Target Device, Information Distribution System, Information Distribution Method, and Non-transitory Computer-Readable medium. US Patent 11,128,480, September 21, 2021.
3. Karakoyunlu, D.; Poo, T.L. Tamper-Resistant Component Networks. US Patent 11,151,290, October 19, 2021.
4. Wentz, C. Systems, Devices, and Methods for Recording a Digitally Signed Assertion Using an Authorization Token. US Patent 11,153,098, October 19, 2021.
5. Herder, C.; Yu, M.; Koushanfar, F. PUFs and Applications: A Tutorial. *Proc. IEEE* **2014**, *102*, 1126–1141.
6. Papakonstantinou, I.; Sklavos, N. *Physical Unclonable Function Design Technologies: Advantages & Tradeoffs. Computer and Network Security*; Daimi, K., Ed.; Spingler International: Edition of 2018; ISBN: 978-3-319-58423-2.
7. Gao, Y.; Ranasinghe, D.; Al-Sarawi, S.; Kavehei, O.; Abbott, D. Emerging physical unclonable functions with nanotechnologies. *IEEE Access* **2016**, *4*, 61–80. https://doi.org/10.1109/ACCESS.2015.2503432. 2016.
8. Jin, Y. Introduction to hardware security. *Electronics* **2015**, *4*, 763–784. https://doi.org/10.3390/electronics 4040763.
9. Rahman, M.; Rahman, F.; Forte, D.; Tehranipoor, M. An aging-resistant ro-puf for reliable key generation. *IEEE Trans*. *ETC*- **2016**.
10. Habib, B.; Kaps, J.; Gaj, K. Efficient SR-Latch PUF. In Proceedings of the ISARC-2015, Bochum, Germany. April 15–17 2015.
11. Holcomb, D.E.; Burleson, W.P.; Fu, K. Power-up SRAM state as an Identifying Fingerprint and Source of TRN. *IEEE Trans. Comp*. **2008**, *57*.
12. Wang, W.; Guin, U.; Singh, A. Aging-Resilient SRAM-based True Random Number Generator for Lightweight Devices" *Journal of Electronic Testing* , v.36 , 2020, https://doi.org/10.1007/s10836-020-05881-6 .
13. Zhang, X.; Jiang, C.; Dai, G.; Zhong, L.; Fang, W.; Gu, K.; Xiao, G.; Ren, S.; Liu, X.; Zou, S. Improved performance of SRAM-based true random number generator by leveraging irradiation exposure. *Sensor* **2020**, *20*, 6132.
14. Chen, A. Comprehensive Assessment of RRAM-based PUF for Hardware Security Applications. IEDM, December 2015.
15. Cambou, B.; Chen, Y.-C. Tamper Sensitive Ternary ReRAM-Based PUF. SAI Computing Conference, London, UK, July 16, 2021.
16. Christensen, T.; Sheets, J. Implementing PUF Utilizing EDRAM Memory Cell Capacitance Variation. US Patent 8,300,450 B2, 30 Oct. 2012.
17. Plusquellic, J.; Bhunia, S. *Systems and Methods for Generating PUF's from Non-Volatile Cells*; WO 20160328578; Nov. 10, 2016.
18. Wang, Y.; Malysa, G.; Wu, S.; Yu, W.-K.; Suh, G.; Kan, E. Flash Memory for Ubiquitous Hardware Security Functions: TRNGs and Device Fingerprints. IEEE Symposium on Security and Privacy, San Francisco, CA, USA, 20-23 May 2012; pp 33–47. doi:10.1109/SP.2012.12.
19. Prabhu, P.; Akel, A.; Grupp, L.; Yu, W.-K.S.; Suh, G.E.; Kan, E.; Swanson, S. Extracting Device Fingerprints from Flash Memory by Exploiting Physical Variations. In Proceedings of the 4th Int. Conf. on Trust and Trustworthy Computing, Pittsburg, PA, USA, June 22-24, 2011.
20. Vatajelu, E.I.; Di Natale, G.; Barbareschi, M.; Torres, L.; Indaco, M.; Prinetto, P. STT-MRAM-Based PUF Architecture exploiting MTJ Fabrication-Induced Variability. *ACM J. Emerg. Technol. Comput. Syst.* **2017**, *13*, 1–21.
21. Zhu, X.; Millendorf, S.; Guo, X.; Jacobson, D.; Lee, K.; Kang, S.; Nowak, M. Physically unclonable function based on programming voltage of magneto-resistive random-access memory, US. Patent 9,343,135, May 17, 2016.
22. Cambou, B.; Orlowski, M. *PUFs Designed with Ternary States*; ACM: New York, NY, USA, 2016; ISBN 978-1-4503-3752-6/16/04.
23. Cambou, B.; Telesca, D. Ternary Computing to Strengthen Cybersecurity, Development of Ternary State based Public Key Exchange. In *SAI Computing Conference*; IEEE, London, UK, July17, 2018.
24. Delvaux, J.; Gu, D.; Schellekens, D.; Verbauwhede. I. Helper Data Algorithms for PUF-Based Key Generation: Overview and Analysis. *IEEE Trans*. *Comput.-Aided Des. Integr. Circuits Syst.* **2015**, *34*, 889–902.
25. Taniguchi, M.; Shiozaki, M.; Kubo, H.; Fujino., T. A stable key generation from PUF responses with a Fuzzy Extractor for cryptographic authentications. In Proceedings of the IEEE 2nd Global Conference on Consumer Electronics (GCCE), Tokyo, Japan, 1–4 October 2013.
26. Kang, H.; Hori, Y.; Katashita, T.; Hagiwara, M.; Iwamura, K. Cryptographic key generation from PUF data using efficient fuzzy extractors. In Proceedings of the 16th International Conference on Advanced Communication Technology, Pyeongchang, Korea， 16–19 February 2014.
27. Boehm, H. Error Correction Coding for PUF: Austrochip, Workshop in Microelectronics, Vienna, Austria, Jan. 1st 2010.
28. Chen, T.; Willems, F.; Maes, R.; Sluis, E.; Selimis, G. A robust SRAM-PUF key generation scheme based on polar codes. *arXiv* **2017**, arXiv:1701.07320 [cs.IT].
29. Maes, R.; Tuyls, P.; Verbauwhede, I. A Soft Decision Helper Data Algorithm for SRAM PUFs. In Proceedings of the 2009 IEEE International Symposium on Information Theory, Seoul, Korea, 28 June–3 July 2009.
30. Cambou, B.; Philabaum, C.; Booher, D.; Telesca, D. Response-Based Cryptographic Methods with Ternary PUFs. FICC, 2019.
31. Cambou, B. Unequally powered Cryptography with PUFs for networks of IoTs. IEEE Spring Simulation Conference, Tucson, AZ, USA, 2019.
32. Cambou, B.; Mohammadi, M.; Philabaum, C.; Booher, D. Statistical Analysis to Optimize the Generation of Cryptographic Keys from PUFs. In Science and Information Conference; Springer: Berlin/Heidelberg, Germany, 2020.
33. Lee, K.; Gowanlock, M.; Cambou. B. SABER-GPU: A Response-Based Cryptography Algorithm for SABER on the GPU. In Proceedings of the 2021 IEEE 26th Pacific Rim Int. Symp. on Dependable Computing (PRDC), Perth, Australia, Dec. 2021.
34. Wright, J.; Fink, Z.; Gowanlock, M.; Philabaum, C.; Donnelly, B.; Cambou, B. A Symmetric Cipher RBC Engine Accelerated Using GPGPU. In Proceedings of the IEEE virtual CNS conference, October 4-6, 2021.
35. NIST - 3rd Round PQC. July 22, 2020. https://csrc.nist.gov/News/2020/pqc-third-round-candidate-announcement.
36. Nejatollahi, H.; Dutt, N.; Ray, S.; Regazzoni, F.; Banerjee, I.; Cammarota. R. Post-Quantum lattice-based cryptography implementations: A survey. *ACM Comput. Surv*. **2019**, *51*, 129. https://doi.org/10.1145/3292548.
37. Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schwabe, P.; Seiler, G.; Stehlé, D. CRYSTALS. https://pq-crystals.org/dilithium. 2019.
38. Nurshamimi, S.; Kamarulhaili. H. NTRU Public-Key cryptosystem and its variants: An overview. *Int. J. Crypto. Res.* **2020**, *10*, 21.
39. D'Anvers, J.-P.; Karmakar, A.; Roy, S.; Vercauteren, F. Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In *International Conference on Cryptology in Africa*; Cryptology ePrint: https://eprint.iacr.org/2018/230.
40. Casanova, A.; Faugere, J.-C.; Macario-Rat, G.; Patarin, J.; Perret, L.; Ryckeghem, J. *GeMSS*, NIST PQC project round 2; Jan. 30th, 2019.
41. Fouque, P.-A.; Hoffstein, J.; Kirchner, P.; Lyubashevsky, V.; Pornin, T.; Prest, T.; Ricosset, T.; Seiler, G.; Whyte, W.; Zhang, Z. *Falcon: Fast-Fourier Lattice-Based Compact Signatures over NTRU*; NIST PQC project round 2; January 30th, 2019.
42. Ding, J.; Chen, M.-S.; Petzoldt, A.; Schmidt, D.; Yang; B.-Y. *Rainbow*; NIST PQC project round 2; Jan. 30th 2019.
43. Maes, R.; van der Leest, V. Countering the Effects of Silicon Aging on SRAM PUFs. HOST, Arlington, VA, USA, 6–7 May 2014 .
44. Grossi, A.; Calligaro, C.; Perez, E.; Schmidt, J.; Teply, F.; Mausolf, T.; Zambelli, C.; Olivo, P.; Wenger, C. Radiation hard design of HfO2 based 1T1R cells and memory arrays. In Proceedings of the 2015 International Conference MEMRISYS, Paphos, Cyprus, 8–10 November 2015.