SAI computing conference 2020

Statistical Analysis to Optimize the Generation of Cryptographic Keys from Physical Unclonable Functions

Bertrand Cambou[1]; Mohammad Mohammadi[1]; Christopher Philabaum[1] and Duane Booher[1]

[1]Northern Arizona University, Flagstaff AZ86011, USA
Bertrand.cambou; mm3845; cp723 ; Duane.booher@nau.edu

Abstract. Physical unclonable functions are not easy to integrate into cryptographic systems because they age, and are sensitive to environmental interferences. Excellent error correcting schemes were developed to handle such drifts, however the computing power needed at the client level can leak information to opponents, and are difficult to deploy to networks of ultra-low power Internet of Things. Response-based cryptography methods, which are server based, use search engines to uncover the erratic keys generated by the physical unclonable functions, minimizing the consumption of electric power at the client level. However, when the defect densities are high, the latencies associated with search engines can be prohibitive. The statistical analysis presented in this paper shows how the fragmentation of the cryptographic keys can significantly reduce the latencies of the search engine, even when error rates are high. The statistical model developed, with Poisson distribution, shows that the level of fragmentation in sub-keys can handle up to 15% error rates. The methodology is generic, and can be applied to any type of physically unclonable functions with defects in the 15% range, or lower.

Keywords: security primitives, internet of things, physical unclonable functions, error correction.

1 Introduction

Physical Unclonable Functions (PUFs) [1-7], have been developed to provide additional layers of protection to cyber physical systems, in particular for access control. They act as the "fingerprints" of microelectronic components, and of the internet of things (IoTs). PUF were successfully introduced with logic gates to protect field programming gate arrays FPGA[8]. Since then, PUFs have been successfully designed with static random access memories (SRAMs) [9], a component available in most electronic devices. However, such solutions are not always tamper resistant; PUFs are designed with Dynamic RAMs [10], flash memory devices [11-12], Resistive RAMs [13-16], and Magnetic RAMs [17-18]. PUFs provide excellent authentication, and access control solutions, without the problem associated with key distribution. They are more difficult to use as cryptographic keys, because PUFs are subject to aging, temperature drifts, electromagnetic interactions, and various environmental effects [19]. Typically, these effects produce 2-10% error rates between the initial readings of the PUFs that are stored as references during enrollment cycles, and the responses generated by these PUFs. The work presented in this work is agnostic about the selection of a particular type of PUF. SRAM PUFs were used because they are widely available, and relatively easy to design.

Error correcting (ECC) algorithms can correct most of these errors, and generate usable cryptographic keys from the PUFs with low false reject rates (FRR), as single-bit mismatches are not acceptable [20-21]. Methods such as the ones using Polar codes have been successfully implemented [22-23]. Other methods such as those using fuzzy extractors are extremely powerful, they can correct PUF responses to generate reliable keys; however some IoT devices may not have enough computing resources to run such codes [24-25]. In several cases, helper data [26] is generated upfront by the server, from error free keys stored in look up tables, to "help" the client device accelerate error correction schemes. The length of the helper data has to be increased when the PUF error rates are high.

Response based cryptographic (RBC) methods have the potential to eliminate the need to use error correction at the client level, as it generates cryptographic keys directly from the un-corrected responses of the PUFs [27-28]. This technology relies on the implementation of efficient search engines, driven by secure servers, which can uncover the keys extracted from PUF responses by the client device. With 256-bit long keys, RBC search engines can uncover keys with up to 1% error rates. The elimination of the weaker cells of a PUF during enrollment cycles [29-30] can reduce the error rates below 10-4, which is low enough for a reliable RBC implementation. It has also been suggested that the fragmentation of the keys into sub-keys can expand the applicability of RBC in the 15% error range rate [31-32]. Key fragmentations can reduce the latencies of the RBC search engine at high error rates, but adds complexity at low error rates. It is therefore desirable to use key fragmentation only when needed.

The objective of the work presented in this paper is to provide a statistical model, and to optimize PUF response based cryptographic schemes at various levels of error rate and fragmentation. Section 2 provides the background information describing error correcting methods, and the response based cryptography needed to be able to exploit PUFs as generators of cryptographic keys. In Section 3, a statistical model of RBC latencies under various conditions is developed. The effect of the level of random defects present in the PUF responses on RBC latencies is analyzed for 256-bit long keys. After presenting methods to fragment these keys into sub-keys, the impact of the level of fragmentation on the RBC latencies is analyzed, with the objective of developing predictive tools. An experimental validation of the suggested models is presented in section 4. The measurements based on commercially available SRAM-based PUFs greatly validate the accuracy of the statistical models when the defect densities are large enough to necessitate RBC searches exceeding 100ms. Finally, we are concluding, presenting how RBC has the potential to become mainstream, to generate defect free cryptographic keys from PUFs for networks of low power IoTs.

2 Background Information

2.1 Error correction methods for PUFs

An example of architecture is shown in Fig. 1. The server downloads the "images" of the PUFs into look up tables during initial enrollment cycles [1]. The term "image" of the PUF is generic, and will have a different meaning for each PUF. In certain cases, the image stored in the server will be a set of challenge-responses describing the PUF. In this study, the image stored in the server is the set of preferential states of each cell of the SRAM memory array, zero or one, after repetitive power-off-power-on cycles. The "handshake", is defined as a set of instructions generated by the server to independently find a particular address in the PUF, and concurrently read the same 256 cells on each side [29]; the keys generated from the PUF can contain errors. Each handshake could point to different addresses, and new keys. Hashing functions and multi-factor authentication are described in [30] to protect the handshake protocol. In the architecture shown in Fig.1, the server generates the helper data [26] from the keys generated with the look up tables, and the ECC engine. The transmission of the helper is often protected by other cryptographic schemes. The client devices correct the keys generated from the PUF, with the helper data, and error correction schemes such as fuzzy extractors [24-25]. The helper data is usually as long as the keys to correct. This method consumes computing power at the terminal level, which may not be available for certain IoT devices. Error correcting methods can also leak information to the opponents through side channel analysis.

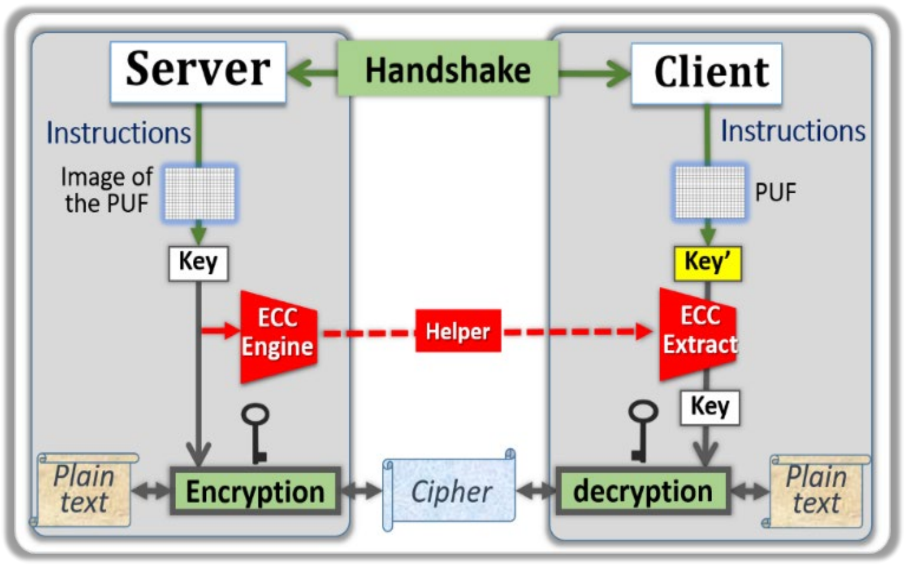


Fig. 1. Example of PUF-based architecture with ECC. The keys generated from look up tables by the server differ from the ones generated from the PUF. The server send helper data to the client device, which uses ECC to retrieve the same key.

2.2 Response based cryptography

An architecture similar to the one presented §2.1, with RBC instead of ECC is shown in Fig. 2. After each handshake, both communicating parties independently generate their keys, the one extracted from the PUF contains errors. The client uses its key **K'** to encrypt a user **ID**, for example with the Advanced Encryption Standard (AES), and send to the server the cipher text **E(ID;K')**. The RBC engine, compares this cipher text with **E(ID;K)**, the cipher text generated by its error free key **K**. Both cipher texts are different unless the two keys are identical. The purpose of the RBC engine, which is described below, is to find in an iterative way the key generating the same cipher text **E(ID;K')**, thereby uncovering **K'** [27].

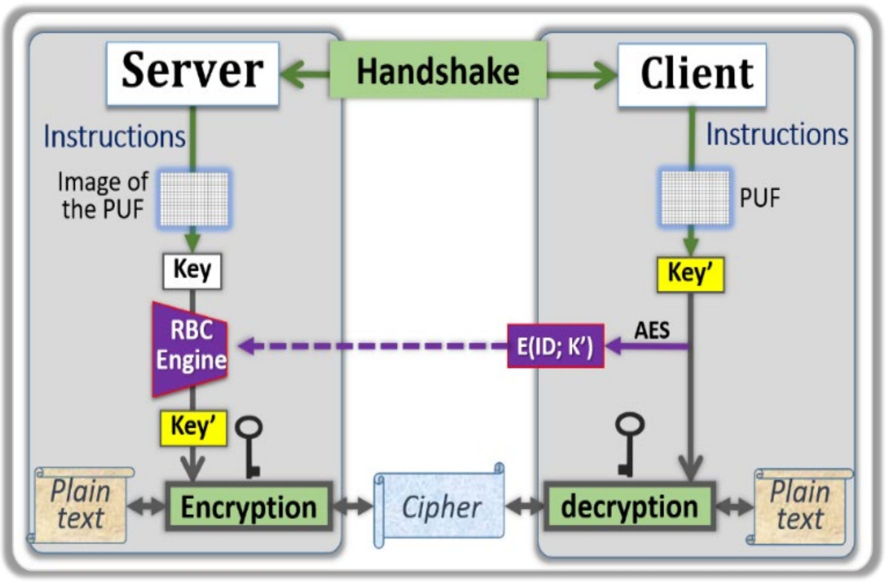


Fig. 2: Example of PUF-based architecture with RBC. The client device encrypt its user ID with its key *K'*. The RBC engine retrieve this erratic key from its key *K*, and *E(ID,K')*.

The starting point of the algorithm used by the RBC search engine, as shown in Fig. 3, is the key **K** extracted from the image of the PUF, and the cipher text **E(ID,K')**, which is openly transmitted by the client device. If the key ***K'*** is at the hamming distance "***a***" of ***K***, the RBC has to test first all possible keys with Hamming distances of ***i*** from ***K***, with ***i*** ∈ {0, ***a***-1} by generating ciphers with each key, and comparing them with ***E(ID,K')***. The server repeats the process within the sphere of Hamming distance ***a***. For 256-bit long keys, the number of possible keys located on this sphere is $\binom{256}{a}$, which is very large when ***a*** is greater than 4. The RBC is limited to PUFs with low defect densities, its latency becomes excessive when ***a*** is greater than 4, for 256-bit long keys, which correspond to a defect density of only 4/256=1.5%. The value of the combination $\binom{256}{4}$ is 1.75 10^8, if each matching cycle takes 1 µs, the RBC latencies are in the three minute range, which is not acceptable. In this protocol, the cipher text **E(ID;K')** is transmitted through unsecure communication channels, however, this is secure with cryptographic protocols such as AES-256. The only way to uncover **K’** is to possess an image of the PUF. Previous work presented how the fragmentation of the keys expands the applicability of RBC to higher defect densities [31-32], however it is desirable to minimize the level of fragmentation to reduce the computing power at the client level; this is the objective of this work. The RBC scheme could be sensitive to potential serious bias due to the lack of robustness of complex networks under attacks [33], which could add errors to the handshakes, and the cipher texts transmitted by the client devices. Mitigation of such attacks should be comprehended in a final implementation of the RBC. The inclusion of redundant correcting schemes is an example of possible remedy.

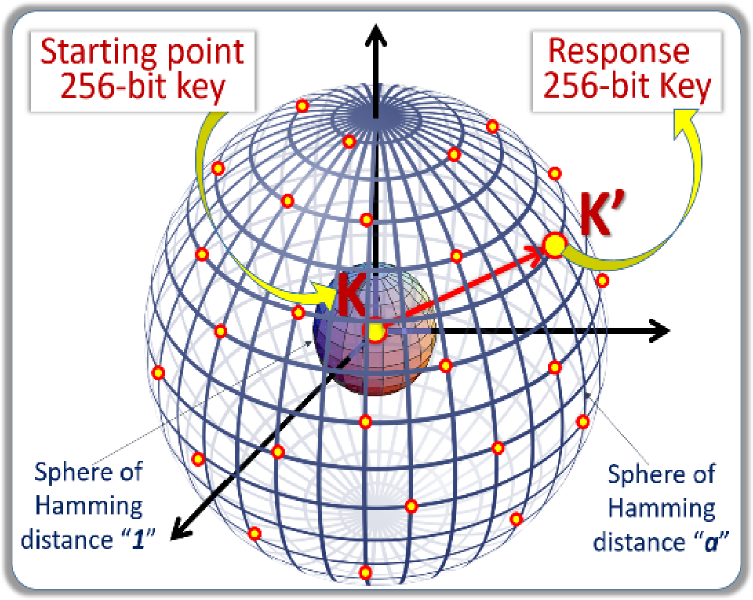

Starting point 256-bit key
Response 256-bit Key
K’
K
Sphere of Hamming distance “1”
Sphere of Hamming distance “a”

Fig. 3: Graphical representation of the RBC engine. The starting point is the key ***K*** generated from the look up table. The iterative search is looking at the key ***K’*** with Hamming distance “***a***” from ***K***, by matching the ciphers.

3 Modeling the latencies

3.1 Effect of random defects on RBC latencies

The RBC matching algorithm compares the cipher text sent by the client device **E(ID,K’)**, with the cipher text computed by the server. The cipher text of the client device is computed with the key **K’** directly generated from the responses of the PUF. The initial cipher text of the server **E(ID,K)**, is computed with the key **K** generated from the image of the PUF that is stored in the look up table as reference. If the two ciphers are different, the server generates the 256 cipher texts from the 256 keys having a Hamming distance of one from the PUF challenges, and compares them to the cipher text transmitted by the client device from the responses.

The process is iterated with keys having higher Hamming distances to find the matching cipher. If the 256-long keys contain exactly X errors, and the latency of one encryption and matching cycle is τ_o , the average latency $L_{(X,1,256)}$ of the RBC search is given by:

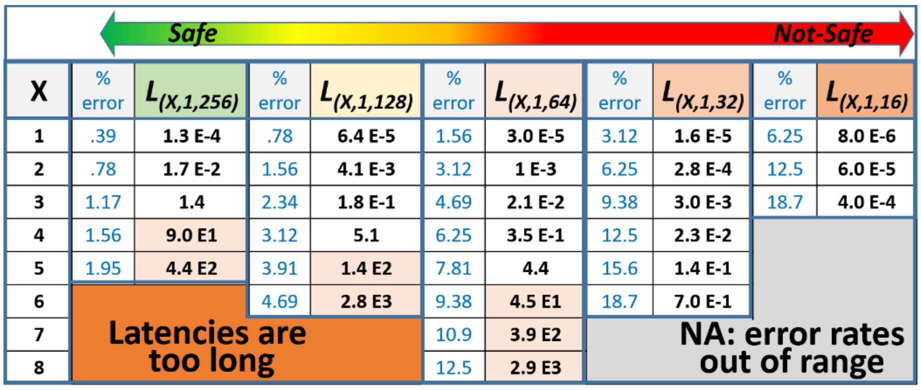
$$L_{(X,1,256)} = \tau_o \left[\sum_{i=0}^{i=a} \binom{256}{i} - \frac{1}{2}\binom{256}{X}\right] \approx \frac{\tau_o}{2}\binom{256}{X} \tag{1}$$

For example if **X** = 4

$$L_{(4,1,256)} = \tau_o + \tau_o\binom{256}{1} + \tau_o\binom{256}{2} + \tau_o\binom{256}{3} + \frac{\tau_o}{2}\binom{256}{4} \tag{2}$$

$$\approx \frac{\tau_o}{2}\binom{256}{4} = 8.74\ 10^7\ \tau_o \tag{3}$$

Using (1), the average RBC latencies $L_{(X,1,256)}$, $L_{(X,1,128)}$, $L_{(X,1,64)}$, $L_{(X,1,32)}$, and $L_{(X,1,16)}$ are computed, and summarized in Fig.4, as a function of the number of error **X** per key, with an assumption that $\tau_o = 1\mu s$. In the experimental work, we use generic AES-256 to generate the cipher texts from 256-bit long cryptographic keys, which has a latency of 500ns with PC powered with quad-I7 chips from Intel.

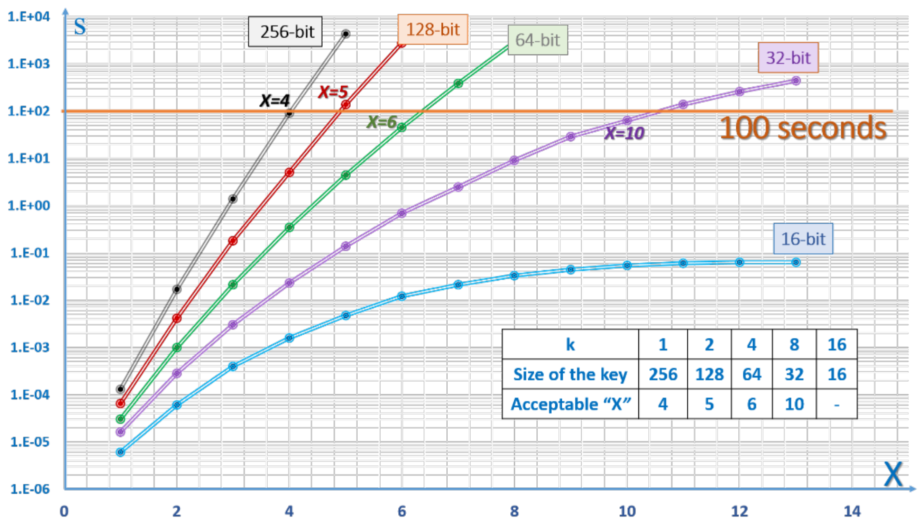

Safe
Not-Safe

X	% error	$L_{(X,1,256)}$	% error	$L_{(X,1,128)}$	% error	$L_{(X,1,64)}$	% error	$L_{(X,1,32)}$	% error	$L_{(X,1,16)}$
1	.39	1.3 E-4	.78	6.4 E-5	1.56	3.0 E-5	3.12	1.6 E-5	6.25	8.0 E-6
2	.78	1.7 E-2	1.56	4.1 E-3	3.12	1 E-3	6.25	2.8 E-4	12.5	6.0 E-5
3	1.17	1.4	2.34	1.8 E-1	4.69	2.1 E-2	9.38	3.0 E-3	18.7	4.0 E-4
4	1.56	9.0 E1	3.12	5.1	6.25	3.5 E-1	12.5	2.3 E-2		
5	1.95	4.4 E2	3.91	1.4 E2	7.81	4.4	15.6	1.4 E-1		
6			4.69	2.8 E3	9.38	4.5 E1	18.7	7.0 E-1		
7	Latencies are too long				10.9	3.9 E2	NA: error rates out of range			
8					12.5	2.9 E3				

Fig. 4: Average RBC latency with various **X**'s, and key size.
With 256 &128-bit long keys, the latencies with X greater than 5.
With 32 &16-bit long keys, the cryptography is exposed to brute force attacks.

The RBC algorithm is efficient when the error rates are small enough. The latencies of the RBC search engines are too slow to process PUFs with error rates higher than 1%, which is the case of most PUFs without other correcting methods. The RBC search fails when the latency exceeds a time limit. In this implementation, the server rejects the authentication of the client device when the latency exceeds 10s, and initiates a new handshake. If the false reject rate FRR at each cycle is below 1%, and three attempts are judged acceptable, the cumulative FRR is below an acceptable 1 part per million (1ppm) level. The maximum acceptable level of defect X for the RBC with 258-bit long keys is 3, which is acceptable with PUF technologies showing low defect rates. The SRAM-based PUFs used in the experimental section of this paper have defect rates below 10^{-4} when the fuzzy cells are removed from the distribution during the enrollment cycle. The RBC scheme in this simple form is not applicable to mainstream SRAM-based PUFs having error rates in the 2-10% range.

Fig. 5 is a set of graphs showing the value of the average RBC latencies $\boldsymbol{L_{(X,1,256/k)}}$ at various numbers of errors $\boldsymbol{X}$ with key lengths of 256/$\boldsymbol{k}$, in which $\boldsymbol{k}$ is respectively equal to 1, 2, 4, 8, and 16. Shorter keys have average RBC latencies that are much lower at equivalent error rates. For example, the RBC searches are always below 100ms with 16-bit long keys. However, these shorter keys are not secure enough against brute force attacks.



k	1	2	4	8	16
Size of the key	256	128	64	32	16
Acceptable "X"	4	5	6	10	-

Fig. 5: RBC latencies $\boldsymbol{L_{(X,1,256/k)}}$ as a function of **X** , and various key length: 256-bit (***k***=1), 128-bit (***k***=2), 64-bit (***k***=4), 32-bit (***k***=8), and 16-bit (***k***=16).

3.2 Poisson distribution for RBC latencies

For a given defect density, the number of bad bits vary key to key. Using Poisson distribution, with a density of error ***D***, and a 256-bit long key, the coefficient λ=256***D*** is used to calculate the probability $\mathbf{P_\lambda(X)}$ to have **X** erratic bit in the key:

$$\mathbf{P_\lambda(X) = e^{-\lambda}} \frac{\lambda^X}{X!} \qquad (4)$$

For example if λ = 4, the probability $\mathbf{P_\lambda(X)}$ versus X (x-axis) is represented in Fig.6:

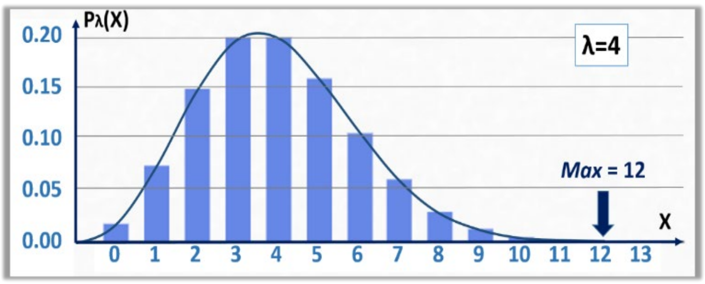


Fig. 6: Poisson distribution $\mathbf{P_\lambda(X)}$ when λ = 4.

When ***Max(λ)*** of ***X*** is defined as the value of X with meaningful probability to occur, the average RBC latency $A_{(\lambda,1,256)}$ for the coefficient λ , and the average defect density ***D***=λ/256 affecting 256-bit long keys is given by:

$$\begin{aligned} A_{(\lambda,1,256)} &= \tau_o \sum_{X=0}^{X=Max(\lambda)} P_\lambda(X) \,.\, L_{(X,1,256)} \\ &= \tau_o \sum_{X=0}^{X=Max(\lambda)} e^{-\lambda} \frac{\lambda^X}{X!} \left[\sum_{i=0}^{i=X} \binom{256}{i} - \frac{1}{2}\binom{256}{X}\right] \\ &\approx \frac{\tau_o}{2} \sum_{X=0}^{X=Max(\lambda)} e^{-\lambda} \frac{\lambda^X}{X!} \binom{256}{X} \end{aligned} \qquad (5)$$

The value ***Max(λ)***, with λ varying from 1 to 6, is shown in Fig.7. The parameter λ is a real number while the value of X, and ***Max(λ)*** are natural integer numbers.

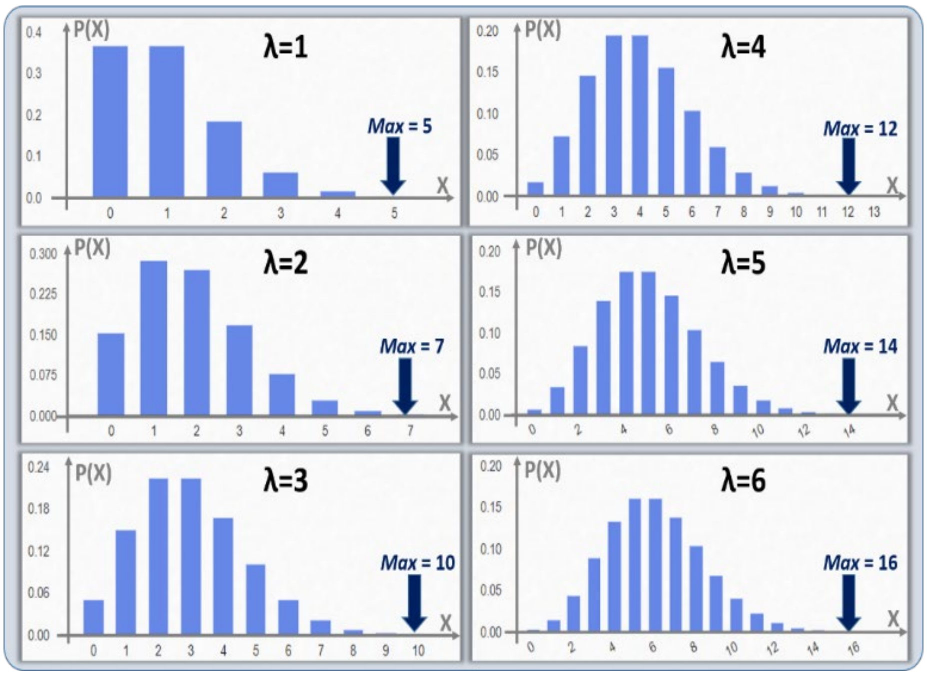


Fig. 7: Distribution $P_\lambda(X)$ with λ = 1, 2, 3, 4, 5, and 6.

4 Fragmentation into sub-keys

To reduce the latencies at higher error rates, we propose the fragmentation of the keys generated by the PUF into sub-keys, also 256-bit long, which are padded with random numbers.

4.1 Padding of the sub-keys

In a fragmentation by two, the first sub-key is generated by keeping the first 128 bits of the 256-bit long key generated by the PUF, filled with a 128-bit long pad containing no errors. The last 128 bits of the PUF, also combined with a 128-bit long pad, generate the second sub-key. Statistically, the two sub-keys show error rates that are half those of full key error rates. In the RBC scheme, the client device sends two cipher texts generated by the two 256-bit long sub-keys. The RBC search engine can process the resulting two ciphers much faster to find the erratic key generated by the PUF. In a fragmentation by k ∈ {**2**, **4**, **8**, **16**}, the first sub-key is generated by keeping the first 256/**k** bits of the key generated by the PUF, filled with a (256-256/**k**)-bit long pad containing no errors. The subsequent ***k*** sub-keys are generated in a similar way. Such a method is shown in Fig.8. The use of padding is widely adopted in many cryptographic schemes to enhance entropy [34]. The padding technology is of prime importance to be able to safely fragment the keys into sub-keys of equivalent strength, however, it is not part of the work presented in this paper, to study the effectiveness of various padding methods.

In the prototype developed here, the padding schemes are secretly shared between the communicating parties during the handshake cycles. The server XORed 512 randomly selected the position of the ternary states of the image of the PUF with the message digest, which is a result of the hashing of a salted random number. The salting technology is implemented with multi-factor authentication. This 512-bit long data stream is needed by the terminal device to eliminate the fuzzy cells of the PUF, and cherry pick the most stable cells of the PUF. The data stream is called a mask. Out of the 512-bit long mask, a 128-bit block is generated to pad the first 128-bit long sub-key, thereby resulting with a full 256-bit long key, in which only 128 bits can be impacted by an erratic PUF. The salted message digest changes during each handshake, and is kept secret. The fragmentation proposed in this paper is agnostic on the preferred padding technology, as long as it is error free, and contains enough entropy.

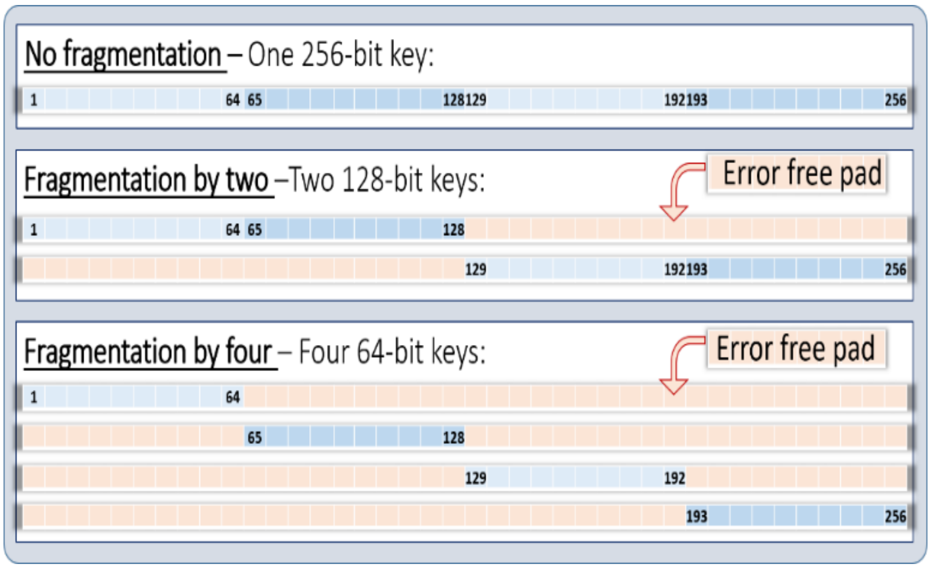

No fragmentation – One 256-bit key:
1
64 65
128129
192193
256
Fragmentation by two –Two 128-bit keys:
Error free pad
1
64 65
128
129
192193
256
Fragmentation by four – Four 64-bit keys:
Error free pad
1
64
65
128
129
192
193
256

Fig.8 : Use of padding for key fragmentation.
The padding use secret information exchanged during the handshake cycles.

4.2 Statistical considerations for the key fragmentation

The parameter $A_{(\lambda,k,256)}$ is defined as the average latency to find the matching keys with RBC, a Poisson coefficient λ, a fragmentation by k sub-keys, and 256-bit long keys. Each sub-key are filled with error free padding to form 256 keys. The average density of error per sub-key is $D' = \frac{1}{k}.D$, with D being the average defect density observed on 256-bit keys before fragmentation. However, the average defect density affecting the fragment of the sub-keys generated by the original key is still **D**. Therefore, the coefficient of Poisson per sub-key is: λ' = λ/k. The corresponding distribution of probability $P_{\lambda'}(X)$ given by (4).

To estimate the average latency $A_{(\lambda,k,256)}$ for a group of e handshakes h with $h \in \{1, e\}$, is assumed that after handshake h, the k sub-keys $K_i)_h$, with $i \in \{1, k\}$, have each a number of errors $h(i)$, which is following the distribution of Poisson with the coefficient λ/k. The RBC latency $\mathcal{T})_h$ of handshake h is given by:

$$\mathcal{T})_h = \sum_{i=1}^{i=k} L_{(h(i),1,256/k)} = \tau_0 \sum_{i=1}^{i=k} \left[\sum_{j=0}^{j=h(i)} \binom{256/k}{j} - \frac{1}{2}\binom{256/k}{h(i)}\right]$$

$$\approx \frac{\tau_0}{2} \sum_{i=1}^{i=k} \left[\sum_{j=0}^{j=h(i)} \binom{256/k}{h(i)}\right] \qquad (6)$$

The average latencies $A_{(\lambda,k,256)}$ for the group of e handshakes are summarized in Table 1, and given by:

$$A_{(\lambda,k,256)} = \frac{1}{e}\sum_{h=1}^{h=e} \mathcal{T})_h = \frac{1}{e}\sum_{h=1}^{h=e} \sum_{i=1}^{i=k} L_{(h(i),1,256/k)}$$

$$= \sum_{i=1}^{i=k} \frac{1}{e}\sum_{h=1}^{h=e} L_{(h(i),1,256/k)} \qquad (7)$$

When e is large enough, the k terms describing the average latency if the sub-keys K_i, with $i \in \{1, k\}$, of (7) are equal, and described by the Poisson distribution of (5):

$$\frac{1}{e}\sum_{h=1}^{h=e} L_{(h(i),1,256/k)} = A_{(\lambda/k,1,256/k)} \qquad (8)$$

Table. 1: Average latency of k sub-keys, after e handshakes.

Hand shake	Sub-key					
	K₁	...	Kᵢ	...	Kₖ	Sum
1	$L_{(1(1),1,256/k)}$	...	$L_{(1(i),1,256/k)}$	...	$L_{(1(k),1,256/k)}$	$\mathcal{T})_1 = \sum_{i=1}^{i=k} L_{(1(i),1,256/k)}$
2	$L_{(2(1),1,256/k)}$	...	$L_{(2(i),1,256/k)}$	...	$L_{(2(k),1,256/k)}$	$\mathcal{T})_2 = \sum_{i=1}^{i=k} L_{(2(i),1,256/k)}$
...	---	...	---	...	---	---
h	$L_{(h(1),1,256/k)}$	...	$L_{(h(i),1,256/k)}$	...	$L_{(h(k),1,256/k)}$	$\mathcal{T})_h = \sum_{i=1}^{i=k} L_{(h(i),1,256/k)}$
...	---	...	---	...	---	---
e	$L_{(e(1),1,256/k)}$	...	$L_{(e(i),1,256/k)}$	...	$L_{(e(k),1,256/k)}$	$\mathcal{T})_e = \sum_{i=1}^{i=k} L_{(e(i),1,256/k)}$
Sum	$\sum_{h=1}^{h=e} L_{(h(1),1,256/k)}$	...	$\sum_{h=1}^{h=e} L_{(h(i),1,256/k)}$	...	$\sum_{h=1}^{h=e} L_{(h(k),1,256/k)}$	$\sum_{h=1}^{h=e} \mathcal{T})_h$
Av.	$A_{(\lambda/k,1,256/k)}$		$A_{(\lambda/k,1,256/k)}$		$A_{(\lambda/k,1,256/k)}$	$A_{(\lambda,k,256)} = kA_{(\lambda/k,1,256/k)}$

With $A_{(\lambda/k,1,256/k)}$ been the RBC search latency of a 256/k bit long key with a coefficient of Poisson λ/k, and without fragmentation. Eq (7) can be written as:

$$A_{(\lambda,k,256)} = k \; . \; A_{(\lambda/k,1,256/k)} \tag{9}$$

It is assumed that **X**= $\boldsymbol{Max(\lambda/k)}$ is the highest number of error per 256/k bit long sub-key, with the meaningful probability to occur, following the distribution $\mathbf{P_{\lambda/k}(X)}$, and with the coefficient $\lambda' = \lambda/k$. The average latency $A_{(\lambda,k,256)}$ can be written as:

$$A_{(\lambda,k,256)} = k\ \tau_0 \sum_{X=0}^{X=Max(\lambda/k)} \mathbf{P_{\lambda/k}(X)}\left[\sum_{i=0}^{i=X}\binom{\frac{256}{k}}{i} - \frac{1}{2}\binom{\frac{256}{k}}{X}\right] \tag{10}$$

With Poisson distribution, this is written as:

$$A_{(\lambda,k,256)} = k\ \tau_0 \sum_{X=0}^{X=Max(\lambda/k)} e^{-\lambda'} \frac{\lambda'^X}{X!}\left[\sum_{i=0}^{i=X}\binom{\frac{256}{k}}{i} - \frac{1}{2}\binom{\frac{256}{k}}{X}\right]$$

$$\approx \frac{k\,\tau_0}{2} \sum_{X=0}^{X=Max(\lambda/k)} e^{-\lambda'} \frac{\lambda'^X}{X!}\binom{256/k}{X} \tag{11}$$

4.3 Effectiveness of the key fragmentation

The effectiveness $\boldsymbol{\eta}$ of a fragmentation by k is given by the ratio of the latency before and after fragmentation:

$$\boldsymbol{\eta} = A_{(\lambda,1,256)} / \, k \; A_{(\lambda/k,1,256/k)} \tag{12}$$

The effectiveness is summarized as follow:

- The average defect densities of each sub-key after fragmentation is $\boldsymbol{D/k}$. The coefficient λ' is $\boldsymbol{k}$ time lower than λ, which pushes the entire Poisson population, including **X**= $\boldsymbol{Max(\lambda/k)}$, toward smaller values.
- The portions of the sub-keys for the RBC search with errors are $\boldsymbol{k}$ times smaller. The RBC latencies are computed with factors proportional to $\binom{256/k}{i}$ versus $\binom{256}{i}$, which are much lower.
- The incorporation of a multiplication by $\boldsymbol{k}$ on (7) and (8) has a small impact, unless the defect densities are extremely small.

4.4 Predictive model

Equations (10) and (12) are appropriate to model RBC average latencies, as a function of the defect density λ, and the level of key fragmentation $\boldsymbol{k}$. The Poisson distribution at various level of defects is summarized in Table 2. The average latency as computed in Fig.5, allows a rough estimate of the expected RBC average latencies:

- Without fragmentation, $\boldsymbol{k}$=1, the RBC probability P(X>3), in color in Fig. 11, is approximately 2% when λ=1; 15% when λ=2; and 40% when λ=3. The latency for P(X=3), see Fig.6, with τ_0=1µs is close to one second. Assuming that the acceptable latency is 10s, and that the location of the defect density follows a Poisson distribution the maximum acceptable defect will be around λ=2, a defect density $\boldsymbol{D}$=2/256 of about 1%.

- With a fragmentation by two, ***k***=2, the RBC latency for P(X=3) is 10 times lower, and the latency for P(X=4) is below ten seconds. The defect density of **λ**=2, is ***D***=2/(256/2)≈1.6%, and the maximum acceptable defect density is about 2%.
- With a fragmentation by four, ***k***=4, the RBC latency for P(X=5) is below ten second. The defect density of **λ**=3, is ***D***=3/(256/4)≈4.6%, and the maximum acceptable defect density is about 6%.
- With a fragmentation by eight, ***k***=8, the RBC latency for P(X=7) is below ten seconds. The defect density of **λ**=5, is ***D***=5/(256/8)≈15.6%, and the maximum acceptable defect density is about 16%.

With a fragmentation by 16, ***k***=16, the RBC latencies stays below ten seconds. The scheme should be able to handle any defect density.

Table. 2: The color code of the table is showing when a given fragmentation results in RBC latencies below 10s; gray for ***k***=1; orange for ***k***=2; purple for ***k***=4; blue for ***k***=8 and 16.

P(X) %	**λ=1**	**λ=2**	**λ=3**	**λ=4**	**λ=5**	**λ=6**
X=0	**37**	**14**	**5**	**2**	**0.6**	**0.2**
X=1	**37**	**27**	**15**	**7**	**3.3**	**1.5**
X=2	**18.4**	**27**	**22**	**15**	**8.4**	**4.5**
X=4	**6**	**18**	**22**	**20**	**14**	**9**
X=4	**1.5**	**9**	**17**	**20**	**17.5**	**13**
X=5	**0.3**	**3.6**	**10**	**16**	**17.5**	**16**
X=6		**1.2**	**5**	**10**	**15**	**16**
X=7		**0.4**	**2.2**	**6**	**10**	**14**
X=8			**0.8**	**3**	**6.5**	**10**
X=9			**0.3**	**1.3**	**3.6**	**6.9**
X=10				**0.5**	**1.8**	**4.1**
X=11				**0.2**	**0.8**	**2.2**
X=12					**0.3**	**1.1**
X=13					**0.1**	**0.5**
X=14						**0.2**

A simplified model is proposed based on (9), to estimate the average RBC latencies for integers λ=1, 2, 3, 4, and 5 , and with k=1, 2, 4, 8, and 16:

$$A_{(\lambda,k,256)} = k \ . \ A_{(\lambda/k,1,256/k)} \approx \ k \ \frac{\tau_0}{2} \binom{256/k}{\lambda} \qquad (13)$$

These estimated RBC latencies $A_{(\lambda,k,256))}$, as a function of the defect densities ***D***=λ/(256/***k***), are plotted in Fig.9.

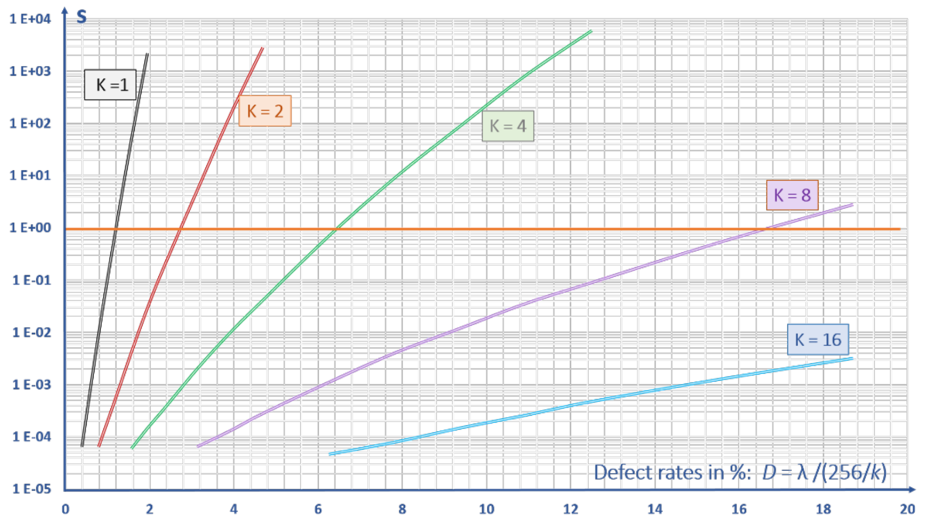


Fig. 9: Modelling the RBC latencies, $A_{(\lambda,k,\,256)} \approx$ $k\,\frac{\tau o}{2}\binom{256/k}{\lambda}$, based on the natural values of λ between 1 and 10. The curves are fitted around these values.

These estimated RBC latencies (y-axis) using eq(11) with Poisson distribution, as a function of the defect densities D=λ/(256/k), (x-axis) are plotted in Fig.10 .

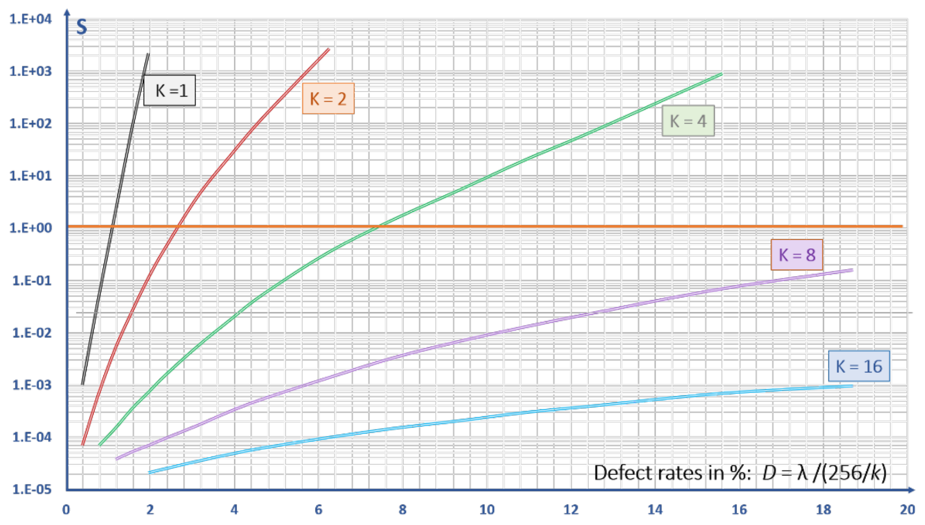


Fig. 10: Modelling the RBC latencies with Poisson distribution and $A_{(\lambda,k,\,256)} = k\ \ A_{(\lambda/k,1,\,256/k)}$.

5 Experimental validation

5.1 Development board

The experimental set up is shown in Fig.11. To validate experimentally the effectiveness of the RBC with fragmentation, commercially available 32-Kbyte SRAM devices from Cypress semiconductor were used as PUFs. During enrollment, the SRAMs are submitted to power-off-power-on cycles, and the resulting patterns are stored in look up tables. About half of the flip-flop of the SRAM cells are mainly responding to such power-off cycles as a "0" state, about half as a "1" state. The error rates of such PUFs are in the 2% to 10% range, due to the cells that have instability in their responses.

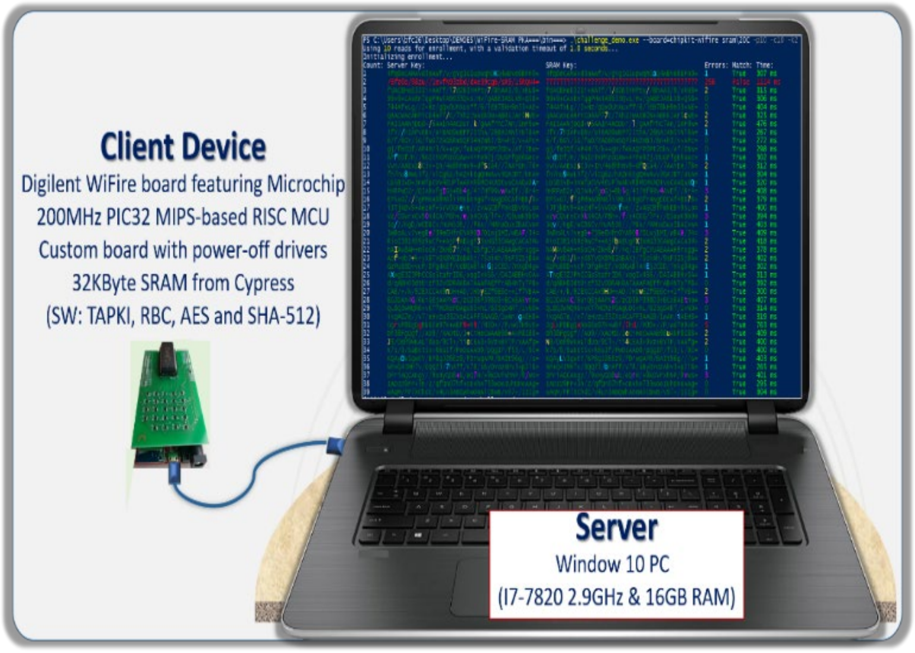

```
PS C:\Users\bfc26\Desktop\DENSES\WiFire-SRAM PKA===\bin===> .\challenge_demo.exe --board=chipkit-wifire sram\20C -p10 -c10 -t2
Using 10 reads for enrollment, with a validation timeout of 1.0 seconds...
Initializing enrollment...
Count: Server Key:                                SRAM Key:                                  Errors: Match: Time:
[illegible]
```
Client Device
Digilent WiFire board featuring Microchip
200MHz PIC32 MIPS-based RISC MCU
Custom board with power-off drivers
32KByte SRAM from Cypress
(SW: TAPKI, RBC, AES and SHA-512)
Server
Window 10 PC
(I7-7820 2.9GHz & 16GB RAM)

Fig. 11: Development board with SRAM-based PUF, and 200MHz RISC MCU. The PC with 2.9GHz quadcore processor performs the RBC searches.

The error rates can be reduced to levels below 10^{-5} by cycling the SRAM multiple times during enrollment, and by eliminating the cells that are not consistent. During 1,000 power-off cycles, about 20% of the cells are showing instability, while the remaining 80% are responding without errors. The error rates of the SRAM PUFs can be adjusted by tracking the position of the cells in the array, and selecting the group of cells with defect densities between 0%, and 20%. The mapping of the cells, and their respective defect density is stored in the server in look up tables. The client device does not memorize any of the enrollment information; it is assumed that the client device can be lost to the enemy. The SRAM PUFs, which are used in this analysis, are not tamper resistant; however, they are appropriate for this experimental work.

WiFire development boards from Digilent, powered by Microchip, were used to drive the SRAM PUFs. These boards contain 200 MHz 32-bit RISC microcontrollers from MIPS, ADC/DAC converters, 2MB flash, and 512KB RAM. The embedded software and cryptographic protocols to extract 256-bit keys from the PUFs were written in C, with software implementation of AES-256, and SHA-256.

On the server side, Window 10 PCs powered by Intel I7 quad core processors were used, they can process an AES cycle in less than one microseconds. The cryptographic protocol is randomly pointing at 256 cells in the PUF, every 2 seconds; the RBC search engine operates with various levels of fragmentation, and PUF defect rates. For each configuration fragmentation-level of defect, the PC typically performed 100 reads with different handshakes and randomly selected 256-bit long keys, and stored the resulting data for analysis.

5.2 Experimental data

To acquire statistically significant data, the RBC latencies are averaged over 100 handshakes, and the PUF error rates vary from 0 to 20% by increment of 1%. For each level of error rate, and new handshake, the server, and the client device independently generate 256-bit long keys. The RBC search engine of the server experimentally measure the latency needed to find a key matching the one generated by the PUF, with fragmentations by 1, 2, 4, 8, and 16. The data used in Fig. 12 was generated with the measurement of 500,000 cells randomly selected, and 10,000 cycles of RBC search.

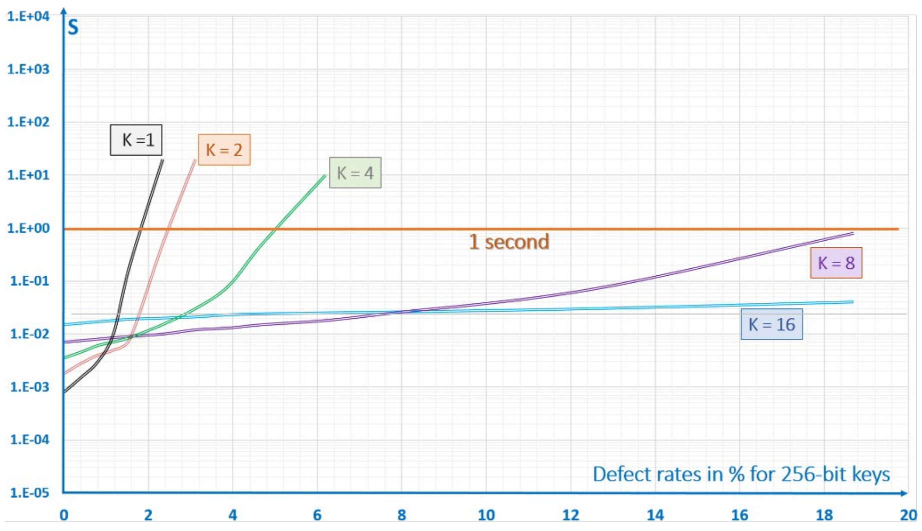


Fig. 12: Experimental measurement of RBC average latencies with 256Kbit SRAM-based PUF, 200MHz RISC MCU development bard, and Window-10 PC with 2.9GHz I7 quad.

The results of the experimental work can be summarized as follow:

- At low defect densities, below 1% rate, the quad-core processor of the PC faces delays due to initialization cycles of 1ms, and the management of the multitasking operations of the PC. The fragmentation increases proportionally with these initialization cycles up to 20ms for ***k***=16. In this range, the RBC is faster without fragmentation, and associated overhead.

- Above defect densities of 1.5%, the fragmentation schemes are needed to keep RBC latencies acceptable. A fragmentation by two is enough to handle defect densities of 2.5%, a fragmentation by four handles 5%, and a fragmentation by 8 handles defect densities higher than 16%.
- The fragmentation by 16 can handle any level of defects within less than 100ms.

The SRAM-based PUFs characterized have defect densities in the 2% to 8% range, therefore a fragmentation by 8 yields consistently reliable RBC searches. With elimination of the erratic cells, and defect densities in the 10^{-5} range, the fragmentation schemes are not needed.

5.3 Comparison between experimental and models

Below 20ms latencies, the model does not take into consideration the response time of the PC; above 10s, the latencies are not acceptable to a normal user case. Therefore, the back-to-back comparison between experimental and modelling data is focusing in the 100ms to 10s range, as shown in Table 3:

Table 3: Defect densities needed to reach a given latency. Experimental versus model.

Defect densities to reach 0.05s to 10s latencies		***k***=1	***k***=2	***k***=4	***k***=8	***k***=16
@0.05s	**Experimental**	1.3%	1.9%	3.5%	11%	
	Simple Model	0.9%	1.9%	4.7%	11.5%	
	Poisson	0.8%	1.7%	4.5%	14.5%	-
@0.1s	**Experimental**	1.4%	2.0%	4.0%	13%	-
	Simple Model	1.0%	2.2%	5.2%	12.8%	-
	Poisson	0.9%	1.9%	5.2%	16.8%	-
@1.0s	**Experimental**	1.7%	2.4%	5.0%	18.5%	-
	Simple Model	1.2%	2.7%	6.3%	16.8%	-
	Poisson	1.2%	2.6%	7.2%	-	-
@10s	**Experimental**	2.2%	3.0%	6.4%	-	-
	Simple Model	1.5%	3.3%	7.7%	-	-
	Poisson	1.4%	3.5%	10%	-	-

- The defect densities needed to reach latencies between 0.1 second and 10 second measured experimentally are roughly similar to the ones anticipated by the model.
- For ***k***=1, about 1% defect rates requires 0.1 second latency for an RBC search, 1.5% requires 1 second, and 2% requires 10 second. The measurements are slightly faster than the model at the same defect density.
- For ***k***=2, about 2% defect rates requires 0.1 second latency for an RBC search, 2.5% requires 1 second, and 3% requires 10 second. The measurements are slightly slower than the model at the same defect density.
- For ***k***=4, about 4% defect rates requires 0.1 second latency for an RBC search, 5% requires 1 second, and 7% requires 10 second. The measurements are slower than the model at the same defect density.
- For ***k***=8, about 13% defect rates requires 0.1 second latency for an RBC search, 18% requires 1 second, and all other defect densities are matched by the RBC engine in less than 10 second. The measurements are slightly faster than the model at the same defect density.
- Both experimental data, and modelling data, show that a fragmentation by 16 always yield successful searches in less than 100ms, regardless of the defect densities.

6 Conclusion and future work

The objectives of the statistical models developed in this work to predict accurately the efficiency of key fragmentation where achieved. The experimental measurements based on SRAM-based PUFs are validating these models, when the latencies are greater than 100ms, to alleviate the inherent latencies of the PC.

The resistance based cryptography is applicable to PUFs with defect densities below 1% (for 256-bit keys), which is the case when fuzzy cells are mapped during enrollment, and removed during key generation. This eliminates the need to use error-correcting methods, helper data, and thereby simplifies PUF-based cryptographic protocols, and enhances security of networks of low power internet of things. Methods to fragment PUF-generated keys enhance the effectiveness of RBC algorithms; a fragmentation by 8 can handle error rates in the 15% range.

The statistical models developed in this work can be used for the following applications:

- Design and optimization of networks of power-constrained IoTs, secured by PUFs. The level of fragmentation is minimized to comprehend the quality of the PUFs used in the system. The level of fragmentation can be coupled with machine learning algorithms to follow the aging of the PUFs, and drifting environments.
- Unequally powered cryptographic systems to protect terminals subjected to variable threats, with noise injection. Noise can be added to the keys generated by the client device to increase the difficulty of the RBC search. High

Performance Computers (HPC) are then used at the server level placing at a disadvantage opponents with inferior computing power.

- Enhancement of the digital signature schemes to secure the blockchain technology. A network of distributed IoTs will use the PUF technology to generate public-private key pairs. Each IoT generate public keys from the keys extracted from the PUFs, and the RBC engine of the certificate authority validate the public keys.

We see the need to complete this research work with various PUFs, which may not have the same distribution of errors as SRAM-based PUFs. The focus is on tamper resistant PUFs to enhance security when the client devices are under side channel analysis. We are studying alternative statistical distribution beside Poisson distribution, which will describe the behavior of various PUFs. We are also conducting experiment with High Performance Computing (HPC), and the implementation of parallel computing to further reduce latencies.

Finally, the deployment of the technology to industry will require the design of custom secure microcontroller, protecting the client devices from side channel analysis. The statistical models enable the development, and optimization of improved RBC schemes mitigating various attacks.

Acknowledgements

The authors are thanking the Information Directorate of the US Air Force Research Laboratory (AFRL) for their support of this research work. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of AFRL. The authors are also thanking the contribution of several graduate students at Northern Arizona University, in particular Vince Rodriguez, and Ian Burke.

References

1. I. Papakonstantinou, and N. Sklavos, “Physical Unclonable Function Design Technologies: Advantages & Trade offs”; Computer and Network Security, editor Kevin Daimi, Spingler, ISBN: 978-3-319-58423-2, 2018;
2. Herder, C., M. Yu, and F. Koushanfar. 2014. "Physical Unclonable Functions and Applications: A Tutorial". Proceedings of the IEEE, vol. 102, no. 8, pp. 1126-1141;
3. R. Pappu, B. Recht, J. Taylor, and N. Gershenfield; Physical one-way functions; Science. Vol 297 No5589 pp2026-2030; 20 Sept 2002;
4. R. Maes and a. I. Verbauwhede, "Physically unclonable functions: a study on the state of the art and future research directions," in Towards Hardware-Intrinsic Security, 2010;
5. Y. Jin; Introduction to hardware security; Electronics 2015, 4, 763-784; doi:10.3390/electronics4040763;

6. Gao, Y., D. Ranasinghe, S. Al-Sarawi, O. Kavehei, and D. Abbott. 2016. “Emerging Physical Unclonable Functions with nanotechnologies”. IEEE, DOI:10.1109/ACCESS.2015.2503432;
7. Rahman, M. T., F. Rahman, D. Forte and M. Tehranipoor. 2016. "An Aging-Resistant RO-PUF for Reliable Key Generation". IEEE Trans. on Emerging Topics in Computing, vol. 4, no. 3;
8. J. Guajardo, S. K. Sandeep, J. S. Geert and T. and Pim; PUFs and PublicKey Crypto for FPGA IP Protection; Field Programmable’07, pp189-195, 2017;
9. D. E. Holcomb, W. P. Burleson, K. Fu; Power-up SRAM state as an Identifying Fingerprint and Source of TRN; IEEE Trans. on Comp., vol 57, No 11; Nov 2008;
10. T. A. Christensen, J. E Sheets II; Implementing PUF utilizing EDRAM memory cell capacitance variation; Patent No.: US 8,300,450 B2; Oct. 30, 2012;
11. J. Plusquellic, and all; Systems and methods for generating PUF’s from non-volatile cells; WO20151056887A1; 2015;
12. Prabhu, P., A. Akel, L. M. Grupp, W-K S. Yu, G. E. Suh, E. Kan, and S. Swanson. 2011. “Extracting Device Fingerprints from Flash Memory by Exploiting Physical Variations”. In 4th international conference on Trust and trustworthy computing;
13. Chen, A. 2015. “Comprehensive Assessment of RRAM-based PUF for Hardware Security Applications”. 978-1-4673-9894-7/15/IEDM IEEE;
14. Cambou, B., F. Afghah, D. Sonderegger, J. Taggart, H. Barnaby and M. Kozicki. 2017. "Ag conductive bridge RAMs for physical unclonable functions”. In 2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, USA;
15. Cambou, B., and M. Orlowski. 2016. “Design of Physical Unclonable Functions with ReRAM and ternary states”. Cyber and Information Security Research Conference, CISR-2016, Oak Ridge, TN, USA;
16. A. Korenda, F. Afghah and B. Cambou, "A Secret Key Generation Scheme for Internet of Things using Ternary-States ReRAM-based Physical Unclonable Functions," International Wireless Communications and Mobile Computing Conference (IWCMC 2018);
17. E. I. Vatajelu, G. Di Natale, M. Barbareschi, L. Torres, M. Indaco, and P. Prinetto; STT-MRAM-Based PUF Architecture exploiting Magnetic Tunnel Junction Fabrication-Induced Variability; ACM trans.; July 2015;
18. X. Zhu, S. Millendorf, X. Guo, D. M. Jacobson, K. Lee, S. H. Kang, M. M. Nowak, D. Fazla; PUFs based on resistivity of MRAM;
19. Becker, G. T., A. Wild and T. Güneysu. 2015. "Security analysis of index-based syndrome coding for PUF-based key generation". In 2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), Washington, DC;
20. Boehm, H. M. 2010. “Error correction coding for physical unclonable functions”. In Austrochip-2010, Workshop in Microelectronics;
21. Maes, R., P. Tuyls and I. Verbauwhede. 2009. "A Soft Decision Helper Data Algorithm for SRAM PUFs". In 2009 IEEE International Symposium on Information Theory;
22. Chen, T. I. B., F. M. Willems, R. Maes, E. v. d. Sluis, and G. Selimis. 2017. "A Robust SRAM-PUF Key Generation Scheme Based on Polar Codes". In arXiv:1701.07320 [cs.IT];
23. Korenda, A., F. Afghah and B. Cambou. 2018. "A Secret Key Generation Scheme for Internet of Things using Ternary-States ReRAM-based Physical Unclonable Functions". In International Wireless Communications and Mobile Computing Conference (IWCMC 2018);

24. Taniguchi, M., M. Shiozaki, H. Kubo and T. Fujino. 2013. "A stable key generation from PUF responses with a Fuzzy Extractor for cryptographic authentications". In IEEE 2nd Global Conference on Consumer Electronics (GCCE), Tokyo, Japan;
25. Kang, H., Y. Hori, T. Katashita, M. Hagiwara and K. Iwamura. 2014. "Cryptographie key generation from PUF data using efficient fuzzy extractors". In 16th International Conference on Advanced Communication Technology, Pyeongchang, Korea;
26. Delvaux, J., D. Gu, D. Schellekens and I. Verbauwhede. 2015. "Helper Data Algorithms for PUF-Based Key Generation: Overview and Analysis". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, no. 6, pp. 889-902;
27. Cambou, C. Philabaum, D. Booher, D. Telesca; "Response-Based Cryptographic Methods with Ternary Physical Unclonable Functions"; 2019 SAI FICC, IEEE; March 2019;
28. B. Cambou, C. Philabaum, D. Duane Booher; Response-based Cryptography with PUFs, NAU case D2018-049, Jun 2018;
29. B.Cambou, P. Flikkema, J. Palmer, D. Telesca, and C. Philabaum; "Can Ternary Computing Improve Information Assurance?"; Cryptography, MDPI; Feb 2018;
30. Cambou, B., and D. Telesca. 2018. "Ternary Computing to Strengthen Information Assurance, Development of Ternary State based public key exchange'. IEEE, SAI-2018, Computing Conference, London, UK;
31. B. Cambou; " Unequally powered Cryptograpgy with PUFs for networks of IoTs"; IEEE Spring Simulation Conference, May 2019;
32. B. Cambou, C. Philabaum, and D. Booher; "Replacing error correction by key fragmentation and search engines to generate error-free cryptographic keys from PUFs", CryptArchi2019, June 2019;
33. Y. Shang; "Subgraph Robustness of Complex Networks under Attacks"; IEEE Tans. On Systems, Man, and Cybernetics: Systems, 2019;
34. European Payments Council, "Guideline on cryptographic algorithms usage and key management", EPC342-08 Nov 2017.