A XOR Data Compiler

Combined with Physical Unclonable Function for True Random Number Generation

Bertrand Cambou
School of Informatics Computing and Cyber Systems
Northern Arizona University
Flagstaff, USA
Bertrand.cambou@nau.edu

***Abstract*—Mathematically generated random number could be weak, while the randomness created physical elements often have insufficient entropy. This paper describes a XOR compiler having the potential to be a true random numbers generator (TRNG), leveraging physical unclonable functions (PUF) designed with memory products. The initial randomness is created by testing only the cells of the memory arrays that are naturally unstable under repetitive measurements, and can easily switch back and forth between a "1" to a"0", thereby generating a random data stream. The level of randomness of this data stream is then enhanced to generate the TRNG. In this paper we are presenting two complementary elements: i) how a fast XOR data compiler, while processing the data available from multiple ternary cells, can create an extremely high level of randomness; and ii) how a combinational probability model allows the quantification of the level of randomness of the TRNG. Deviations of absolute randomness of these TRNG in terms of probability to be non-random can be lower than 10^{-10} which is accepted as non-detectable for existing and computers of the foreseeable future.**

Keywords—true random number generators (TRNG); physical unclonable function (PUF); exclusive OR logic (XOR); memory arrays component; ternary states

I. INTRODUCTION – BACKGROUND INFORMATION

Pseudo random number generators (PRNG), and true random number generators (TRNG) are mainstream to strengthen cryptographic protocols, see reference [1-7]. The randomness based on mathematical algorithms of PRNGs could be weak when the mathematical algorithms are known by a crypto-analyst. Such algorithms can also consume significant amounts of computing power, and clock cycles which may be a problem for small internet of things (IoT) peripherals.

The use of physical elements such as physical Unclonable functions (PUF) as a source of natural randomness have been adopted for the design of true random number generators (TRNG). But these solutions have also their own set of limitations. The randomness of the physical elements can vary when subjected to temperature changes, aging, electromagnetic interferences, and other parametric drifts.

The PUFs, regardless of their design, have to exhibit enough predictability overtime to be used as cryptographic primitives for reliable authentications, see reference [8-18]. The reference patterns of the PUFs that are generated up front during the setup of protocol, called challenges, have to be compared over the life of the component with the patterns, called responses, generated during the authentication cycles, thereby resulting in low challenge-response-pair (CRP) error rates. The PUF challenges should act as predictable "digital fingerprints" of the component, while the responses should be easily recognizable as a measurement of the same "digital fingerprints". PUFs exploit the device to device randomness that is created during the manufacturing process of micro-components, they should all be statistically distinct from each other's.

This property of predictability of the PUFs is not necessarily conducive to the design of quality TRNGs which rely on physical randomness to generate a fresh random number at every query. The main objective of this work is to design an architecture that can achieve the dual objective of a PUF with low CRP error rate, together with highly random TRNG, and this at low cost when used by high volume IoTs. The quantification of the resulting level of randomness of the TRNG is also an important consideration.

A. Memory with ternary states

The methods to design PUFs and TRNGs with memories are very diverse, and have been widely published with SRAM [19-22], Flash RAMs [23], DRAMs [24], magnetic RAMs [25, 26], resistive RAMs [27], or other solid state drives [28]. The cryptographic protocols leveraging memory PUFs are distinct than the ones developed with other mainstream PUFs such as ring oscillators, or gate delays. It is indeed possible to program and read the content stored in random access memories, but unlike it is done in a content addressable memory (CAM), it is not possible to compare within the memory arrays the challenges and the responses for authentication purpose. The CRP matching has to be done outside the memories after extraction of the responses together with error correction methods. The cell to cell physical parameter variations due to normal manufacturing variations are such that a certain number of cells within the memory arrays behave in a way that is too erratic for CRP generation, fortunately these are the cells that can be exploited to generate highly random numbers.

The concept of memory based PUFs with ternary states having random number generation capabilities was described previously [29-30]. In these publications the cells of a memory array are segmented into three states, between the predictable cells which yield a reproducible "0" or "1" when subject to repetitive tests, and the ternary "X" cells that are unstable, and can randomly switch back and forward at each event. The PUF

978-1-5090-5443-5/17/$31.00 ©2017 IEEE

challenge-response-pairs can be generated from the cells having predictable "0"s and "1"s. It was suggested in these publications, and this based on experimental data, that when the criteria to sort out the solid cells is more stringent, which mean that a relatively larger proportion of the cells are given the state "X", the CRP error rate can be extremely low.

Conversely the X-cells can be exploited to generate streams of bits with random numbers. In such case, the criteria to sort out these X-cells within the array, which are selected for the generation of TRNG, is to have almost equal probability to test these cells either as a"0" or a "1" every time they are tested. Most advanced memory designs incorporate the built-in-self-test (BIST) feature [31] that can perform repetitive testing of the memory arrays. BISTs are currently used in manufacturing to sort out the good memory parts during wafer probe. The same BIST features can be applied to sort out the X-cells in memory arrays for TRNGs.

In the references [29-30] it was suggested that additional confusion and scrambling need to be created to the streams of bits generated from the X-cells to obtain real quality TRNG. The XOR data compiler described in this paper is intended to provide additional randomness, and reach the level of entropy expected in final TRNGs.

One of the complicated requirement of any RNG, or TRNG, is the need to quantify the figure merit of randomness of the RNG, and the entropy. Each bit "x" of a data stream of N bits should have precisely a 50% probability to be either a "1" or a "0" to be part of a perfect TRNG:

$$P_{(X=1)} = P_{(X=0)} = 0.5 \quad (1)$$

The average deviation from perfect randomness λ is given by:

$$\lambda = | 0.5 - P_{(X=1)} | = | 0.5 - P_{(X=0)} | \quad (2)$$

Assuming that the length of the RNG data stream is $N=128$, with $P_{(X=0)} = 0.5$, the number of possible combinations, also called entropy, is $2^{128} = 3.4\ 10^{38}$, which is large enough to protect the cryptographic functions from existing or foreseeable computers. When the RNG is not totally random, in this case $\lambda \neq 0$, the entropy is lower than 2^{128}, and is further reduced with larger λ.

A position paper from the National Institute of Standard and Technology (NIST) [32] suggested in 1999 that λ greater than 10^{-3} would not be acceptable, because sophisticated crypto-analysis methods could then be effective. This was revisited by NIST in 2010 and others [33-34]. The value of λ that is considered as acceptable to get a safe TRNG is a moving target as modern computers are increasingly powerful. To the best of our knowledge, $\lambda < 10^{-5}$ is currently considered an excellent target, while $\lambda < 10^{-10}$ is considered outstanding.

B. XOR for randomization and cryptography

Exclusive OR, XOR, is a Boolean logic gate widely adopted in cryptography. Two input bits X, and Y are transformed into $Z = X \oplus Y$, with the following truth table:

$$Z=0 \text{ if } X=Y \text{ (0 or 1)}, Z=1 \text{ if } X \neq Y \quad (3)$$

XOR logic is part of most encryption algorithms such as the Data Encryption System (DES), the Advanced Encryption System (AES), and hash functions such as SHA. XOR functions are adding confusion and randomization in the encryption process while been reversible in the decryption process. As part of the encryption, the data streams generated by the plain texts are often XORed with cryptographic keys, or sub-keys, then XORed again during decryption.

XOR scramblers are used to enhance randomization in multicarrier communications [35]. XOR are used to generate scrambling sequences to achieve data randomization in a memory circuit, as well as enhancing random number generators [36-37]. Some of the important reasons for the use of XOR functions in cryptography are:

- Z is not disclosing the value of X and Y:

  $Z=0$ can be the result of the pair 00, or the pair 11;

  $Z=1$ can be the result of the pair 01, or the pair 10;

- XOR is a symmetrical function when applied twice:

$$X \oplus Y \oplus Y = X \quad (4)$$

- If two bits X and Y are random, the bit Z, defined by $Z = X \oplus Y$, is even more random than either X or Y.

These properties are exploited in the design of the XOR data compiler as presented §II.

II. DESCRIPTION OF THE XOR COMPILER

A. Sorting out the ternary states "X"

The selection of the cells that are somewhat unstable, and referred to as ternary state "X", are the ones that can randomly switch between a "0" and a "1", and this based on the determination that their parameter $\mathcal{P}$ is either below (for a "0"), or above (for a "1") of an arbitrary threshold value $T1$ of parameter $\mathcal{P}$ which can be the median value of the distribution.

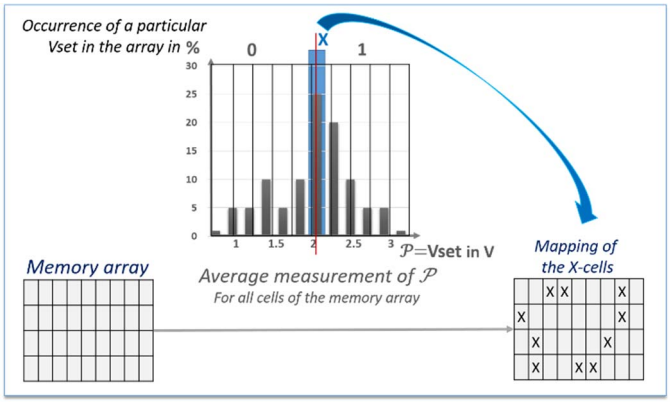

Fig. 1. Mapping of the X-cell within a memory array

The process of selecting the X-cells is represented in Fig. 1. Let us assume that within the cells of the memory array the distribution of the physical parameter $\mathcal{P}$ that is used to determine if a cell is a "0" or a "1" is following a normal distribution, with a standard variation σ_{Array} due to cell to cell variations created during manufacturing, and other instabilities.

978-1-5090-5443-5/17/$31.00 ©2017 IEEE

Let us also assume that every time a new measurement of $\mathcal{P}$ is done on the same cell, this measurement will also follow a normal distribution with a standard variation σ_{Cell} responding to various measurement instabilities, noise, and environmental variations. σ_{Cell} should be lower than σ_{Array} for commercial memories to be sure that "0' and "1" are clearly defined.

The segmentation of the cells between the one that are unstable, the X-cells, and the more stable ones, can be such that only the cells extremely close to the transition of parameter $\mathcal{P}$ between "0' and "1" can be selected. The average value $\boldsymbol{Tx}$ of parameter $\mathcal{P}$ of each X-cells should be such that:

$$|Tx - T1| \ll \sigma_{Cell} \quad (5)$$

This maximizes the chance of a random flip between "0s" and "1s". In order to enhance the level of randomness only a very small percentage of the memory arrays are then selected. For 128 bits RNG, as much as 10,000 bits of the array could be required for high randomness. Considering that current secure micro-controllers have very large embedded memory density, typically in the 1 to 100 Mbits, the percentage of the array consumed for TRNG is extremely small.

In the following sections of this paper we are studying how to enhance the randomness generated by extracting a data stream from the X-cell. One of the tradeoffs to model is the compromise between tightly selecting the X-cell around the threshold $\boldsymbol{T1}$ thereby reducing the need to further reduce non-randomness with the XOR data compiler, versus enlarging the selection criteria of the X-cell, and increasing the reliance on the XOR data compiler to get to the desired level of randomness.

B. Description of the cells with ternary states

As shown in Fig 2, the cells that are sorted as unstable with an "X" state can be segmented into two subgroups:

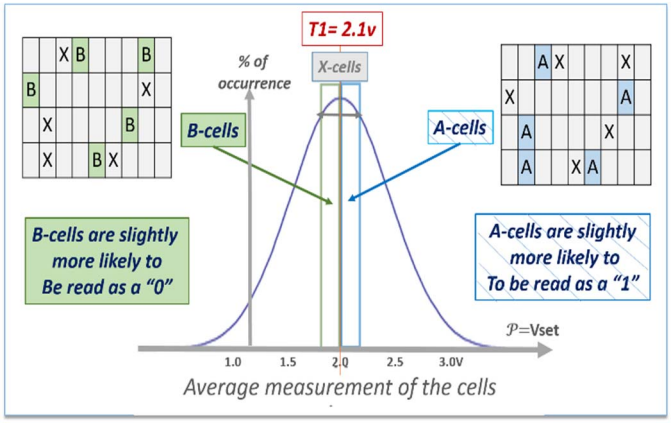

Fig. 2. Sorting out the X-cell into A and B groups

- The cells that have a higher probability to be tested as a "1" are called A-cells, see Fig 3. They have an average probability $\boldsymbol{P_A}$ to be tested as "1" $\boldsymbol{P_A = .5 + \lambda_A}$, and an average probability $\boldsymbol{P'_A}$ to be a "0" $\boldsymbol{P'_A = .5 - \lambda_A}$;

$$\boldsymbol{1 = P'_A + P_A} \quad (6)$$

- The cells that have a higher probability to be tested as a "0" are called B-cells, see Fig 4.0. They have an average probability $\boldsymbol{P'_B}$ to be a "0" $\boldsymbol{P'_B = .5 + \lambda_B}$, and an average probability $\boldsymbol{P_B}$ to be a "1" $\boldsymbol{P_B = .5 - \lambda_B}$;

$$\boldsymbol{1 = P'_B + P_B} \quad (7)$$

The selection of the transition of parameter $\mathcal{P}$ between "0s" and "1s" can be such that the number of A-cells equal the number of B-cells and that $\boldsymbol{P_A = P'_B}$, or $\boldsymbol{\lambda_A = \lambda_B}$.

As it is presented in §3, based on experimental data, with about 2% of the cell population selected as X-cell, $\boldsymbol{P_A}$ and $\boldsymbol{P'_B}$ are quite close 0.5 with $\boldsymbol{\lambda_A = \lambda_B}$ in the $\boldsymbol{2\ 10^{-2}}$ range. This is far from the level of randomness necessary to generate quality TRNG, this based on NIST criteria. In the next § it is shown how the XOR compiler can provide additional enhancement in the level of randomness required to design quality TRNG.

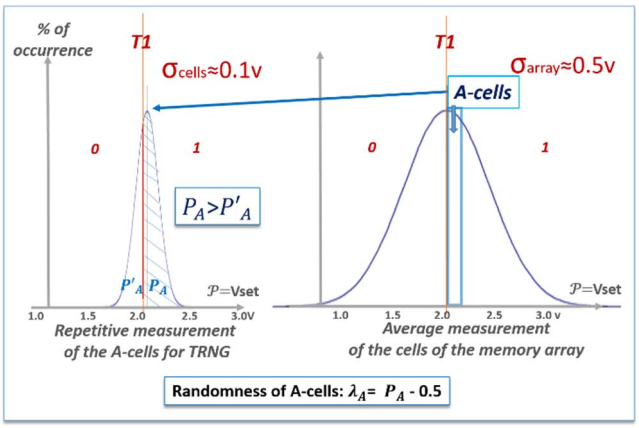

Fig. 3. A-cell with higher probability to be tested

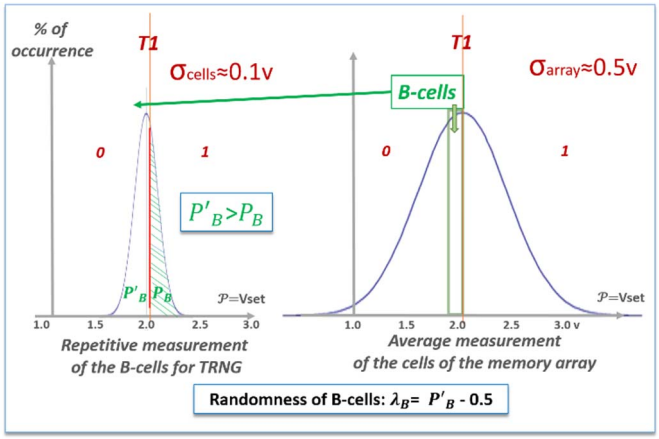

Fig. 4. B-cell with higher probability to be tested "0"

C. Use of a XOR data compiler and ternary states

Let us assume that the stream of numbers generated by a memory array with ternary states is: $\boldsymbol{\{a_1, a_2, \ldots, a_i, \ldots a_n\}}$.

This stream has to be grouped in chunks of $\boldsymbol{f}$ bits:

$\boldsymbol{\{a_{j1}, a_{j2}, \ldots, a_{ij}, \ldots, a_{jf}\}}$ with $\boldsymbol{f<n}$.

For example, 1,280 random bits can be organized in 128 chunks of 10 bits. The randomization is achieved with a XOR data compiler to create a stream of true random numbers, see Fig 5.0: $\boldsymbol{\{c_1, c_2, \ldots, c_j, \ldots, c_m\}}$

978-1-5090-5443-5/17/$31.00 ©2017 IEEE

This stream is defined as follow:

$$c_j = a_{j1} \oplus a_{j2} \oplus \dots \oplus a_{ij} \oplus \dots \oplus a_{jf} \text{ and } n = m.f \quad (8)$$

Such XOR compiler can be implemented at the software level, or in hardware, as only a few logic gates are necessary. These gates can be inserted in the state machine of the memory device to directly feed the secure processor, and crypto-processor with quality TRNG. Unlike other random number generation that are sequential (the random number i of a stream of N bits is often generated with the previous random numbers of the stream), the XOR computations can be done in parallel. XOR computation is a congruent modulo 2 without carry over:

$$C_j = a_{j1} \oplus a_{j2} \oplus \dots \oplus a_{jf} = (a_{j1} + a_{j2} + \dots + a_{jf}) \bmod 2 \quad (9)$$

A faster way to compute the XOR operation is to count how many "1s" are presents in the stream $\{a_{j1}, a_{j2}, .., a_{jf}\}$. If the number of "1s" in the stream is odd then $c_j = 1$, when even $c_j = 0$. The quantification of the effect of the XOR compiler in the randomization of a data stream is presented in the next §.

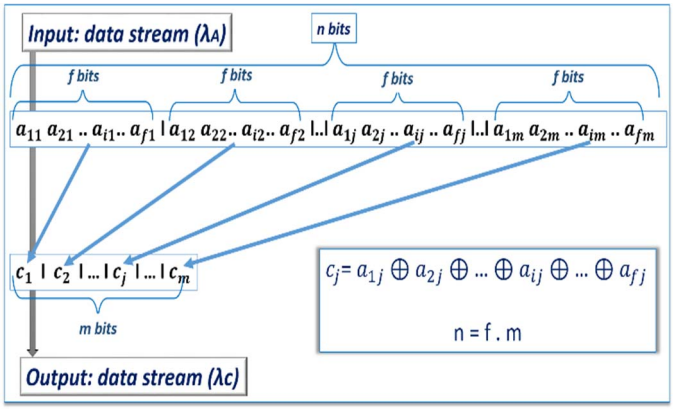

Fig. 5. Description of the XOR compiler

D. Probabilistic model

The objective is to model the probabilities P_C for c_j, $c_j = a_{j1} \oplus a_{j2} \oplus \dots \oplus a_{jf}$, to be a "1", and the variation λ_C from randomness. Each of the random bits that are from the chunk of bits $\{a_{j1}, a_{j2}, \dots, a_{jf}\}$ are either generated from A-cells, or B-cell. Any of these chunks are f long will have s A-cells and t B-cells with $s+t=f$. Considering that the probability to have an A-cell, and a B-cell is equal to 0.5, the probability to have such a (s, t) configuration is: $C_{f,s} / f!$ with $C_{f,s} = f!/s!(f-s)!$.

<u>*Set-0: all cells generating $\{a_{j1}, a_{j2}, \dots, a_{jf}\}$ are A-cells.*</u>

In this case $t=0$, and $s=f$. The probability of any of the f cells of the stream to be a "1" is P_A, and the probability to be a "0"is P'_A. The Bernoulli formula is giving the following polynomial:

$$1 = \sum_{i=0}^{i=f} C_{f,i} \; P_A^i P'^{f-i}_A \quad \text{with} \quad C_{f,i} = f!/i!(f-i)! \quad (10)$$

$$= \sum_{i=0}^{i=f} [i \bmod 2] \; C_{f,i} \; P_A^i P'^{f-i}_A \quad \leftarrow P_c \quad (11)$$

$$+ \sum_{i=0}^{i=f} [(i+1) \bmod 2] \; C_{f,i} \; P_A^i P'^{f-i}_A \quad \leftarrow P'_c \quad (12)$$

The term $P_A^i P'^{f-i}_A$ of eq. (9), (10), and (11) corresponds to a configuration where i bits are "1s", and $f-i$ bits are "0".

When i is odd, $i \bmod 2 = 1$, the resulting XOR $c_j = 1$, and P_C is:

$$P_c = \sum_{i=0}^{i=f} [i \bmod 2] \; C_{f,i} \; P_A^i P'^{f-i}_A \quad (13)$$

When i is even, $(i+1) \bmod 2 = 1$, $c_j = 0$, and P'_C is:

$$P'_c = \sum_{i=0}^{i=f} [(i+1) \bmod 2] \; C_{f,i} \; P_A^i P'^{f-i}_A \quad (14)$$

Eq. (12) and (13) can be written in a different way while taking into consideration whether f is even or odd.

- If f is even, $f=2k$:

$$P_C = \sum_{i=0}^{i=k-1} C_{2k,2i+1} \; P_A^{2i+1} P'^{2k-2i-1}_A \quad (15)$$

$$P'_C = \sum_{i=0}^{i=k} C_{2k,2i} \; P_A^{2i} P'^{2k-2i}_A \quad (16)$$

Eq. (12) and (14) are equivalent, while eq. (13) and (15) are also equivalent. In this case $P_C < P'_C$ and can be written as $P_C = 0.5 - \lambda c_j$ with λc_j the deviation from pure randomness which is $P_C = 0.5$. It can also be written $P'_C = 0.5 + \lambda c_j$.

- If f is odd, $f=2k+1$:

$$P_C = \sum_{i=0}^{i=k} C_{2k+1,2i+1} \; P_A^{2i+1} P'^{2k-2i}_A \quad (17)$$

$$P'_C = \sum_{i=0}^{i=k} C_{2k+1,2i} \; P_A^{2i} P'^{2k+1-2i}_A \quad (18)$$

Eq. (13) and (17) are equivalent, while eq. (14) and (18) are also equivalent. In this case $P_C > P'_C$ and can be written as $P_C = 0.5 + \lambda c_j$ with λc_j the deviation from pure randomness. It can also be written $P'_C = 0.5 - \lambda c_j$.

<u>*Set-1: $\{a_{j1}, a_{j2}, \dots, a_{jf}\}$ are generated by A-cells & B-cells*</u>

In this case the f cells can randomly be generated by A-cells and B-cells. The symmetry between the A-cell and the B-cells ($P_A = P'_B$ and $P'_A = P_B$) result in the following property:

- If the chunk of bits $\{a_{j1}, a_{j2}, \dots, a_{jf}\}$ is generated by an even number of B-cells, the probabilities P_C and P'_C are the same as if the chunk was only generated by A-cells as described in 3-2-2:

If f is even, P_C and P'_C are computed with eq. (15) and (16); if f is odd, P_C and P'_C are computed with eq. (17) and (18).

- If the chunk of bits $\{a_{j1}, a_{j2}, \dots, a_{jf}\}$ is generated by an odd number of B-cells, the probabilities P_C and P'_C are the opposite than if the chunk was only generated by A-cells as described by eq. (15) to eq (18):

If f is even, Pc is (eq.(16)): $P_C = \sum_{i=0}^{i=k} C_{2k,2i} \; P_A^{2i} P'^{2k-2i}_A$, and P'c is (eq.(15)): $P'_C = \sum_{i=0}^{i=k} C_{2k,2i} \; P_A^{2i} P'^{2k-2i}_A$;

if f is odd, Pc is (eq.(18)): $P_C = \sum_{i=0}^{i=k} C_{2k+1,2i} \; P_A^{2i} P'^{2k+1-2i}_A$, and P'c is (eq.(17)): $P'_C = \sum_{i=0}^{i=k} C_{2k+1,2i+1} \; P_A^{2i+1} P'^{2k-2i}_A$

Set-1 looks complicated, however the purpose of this model is to calculate the absolute deviation from perfect randomness, it is not really important to know if $P_C > P'_C$, or if $P'_C > P_C$. In all cases $|\lambda c_j|$ is the statistical deviation from pure randomness, regardless of P_C being greater or lower than P'_C. We are making the assumption that considering all cells as A-cells is to simplifying the computation without reducing the

978-1-5090-5443-5/17/$31.00 ©2017 IEEE

accuracy of model. We will verify this statement experimentally.

E. Checking the model: XOR of two cells

The simplest way to implement the method is to XOR the data stream two bits by two bits for the entire stream, ***f=2***. There are three configurations, both cells are A-cell, one cell is an A-cell and the second is a B-cell, and both cells are B-cells.

Let us choose: $P_A=P'_B=0.52$; $P'_A=P_B=0.48$; $\lambda_B=\lambda_A=2\ 10^{-2}$.

Set-0: Number of A-cells is even: two A-cells, or two B-cells.

- The probability P'_C to have $c_j=a_{j1}\oplus a_{j2}$ been a "0" is:

$$P'_C = P'^2_A + P^2_A = 0.5008 \rightarrow \lambda_C = 8\ 10^{-4} \quad (19)$$

(two cases to get c_j at "0": {00} or {11})

- The probability P_C to have $c_j=a_{j1}\oplus a_{j2}$ been a "1" is:

$$P_C = 2\,(P_A P'_A) = .4992 \quad (20)$$

(two cases to get c_j at "1": {01} or {10})

Set-1: Number of A-cells is odd: one A-cells, and one B-cell.

- The probability P_C to have $c_j=a_{j1}\oplus a_{j2}$ been a "1" is:

$$P'_C = P^2_A + P'^2_A = 0.5008 \rightarrow \lambda_C = 8\ 10^{-4} \quad (21)$$

(two cases to get c_j at "1": {10} or {01})

- The probability P'_C to have $c_j=a_{j1}\oplus a_{j2}$ been a "0" is:

$$P_C = 2\,(P_A P'_A) = .4992 \quad (22)$$

(two cases to get c_j at "1": {00} or {11})

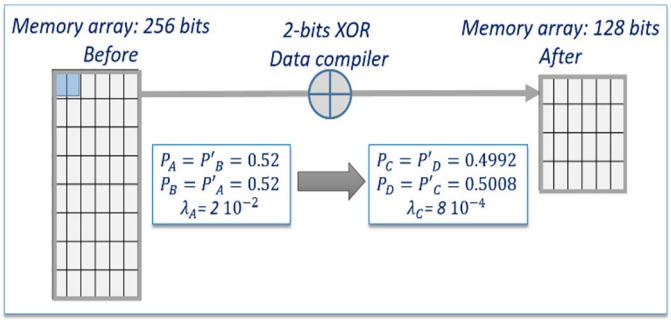

Fig. 6. Example 2-bits XOR compiler

Assuming that we are starting with a stream of 256 bits having 50% A-cells, and 50% B-cells with $\lambda_A=2\ 10^{-2}$, the 2-bit XOR compiler can statistically generate a stream of 128 bits having 50% C-cells, and 50% D-cells with $\lambda_C=8\ 10^{-4}$, see Fig. 6.0. The C-cells are made of pairs of either AA cells or BB cells (Set-0). The D-cells are made of pairs of either AB cells or BA cells (Set-1).

In both Set-0 and Set-1, ***f=2*** is even. The general equations developed in §2-4 , eq. (15) to (18) are applicable. When the number of B-cell is even $P_C < P'_C$, and are reversed when the number of B-cell is odd $P_C > P'_C$. The level of non-randomness $\lambda_C=8\ 10^{-4}$ is **25** times smaller than the level of non-randomness before the XOR compiler, $\lambda_A=2\ 10^{-2}$.

F. Checking the model: XOR of three cells

In this case the data steam is XOR'ed three bits at a time for the entire stream, ***f=3***. There are four configurations, all cells are A-cell, two cells are A-cell & one cell is B-cell, one cell is A-cell & two cells are B-cells, and all cells are B-cell.

Let us choose: $P_A=P'_B=0.52$; $P'_A=P_B=0.48$; $\lambda_B=\lambda_A=2\ 10^{-2}$.

Set-0: B-cells are even: three A-cells, A-cell & two B-cells.

- The probability P_C to have c_j been a "1" is:

$$P_C = P^3_A + 3\,P_A P'^2_A = 0.500032 \rightarrow \lambda_C = 3.2\ 10^{-5} \quad (23)$$

(four cases to get $c_j=1$: {111}, {100}, {010}, or {001})

- The probability P'_C to have $c_j=0$ is:

$$P'_C = P'^3_A + 3\,P'_A P^2_A = 0.499968 \quad (24)$$

(four cases to get $c_j=0$: {011}, {101}, {110}, or {000})

Set-1: B-cells are odd: Three B-cells, B-cell & two A-cells.

- The probability P'_C to have $c_j=0$ is:

$$P'_C = P^3_A + 3\,P_A P'^2_A = 0.500032 \rightarrow \lambda_C = 3.2\ 10^{-5} \quad (25)$$

(four cases to get $c_j=0$: {000}, {110}, {011}, or {101})

- The probability P_C to have $c_j=1$ is:

$$P_C = P'^3_A + 3\,P'_A P^2_A = 0.499968 \quad (26)$$

(four cases to get $c_j=1$: {111}, {100}, {010}, or {001})

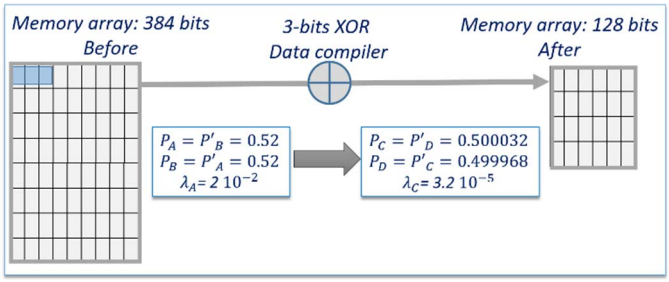

Fig. 7. Example of 3-bits XOR compiler

Assuming that we are starting with a stream of 384 bits having 50% A-cells, and 50% B-cells with $\lambda_A=2\ 10^{-2}$, the 3-bit XOR compiler can statistically generate a stream of 128 bits having 50% C-cells, and 50% D-cells with $\lambda_C=3.2\ 10^{-5}$, see Fig. 7.0. The C-cells are made of triplets of either AAA cells, ABB cells BAB cells, or BBA cells (Set-0). The D-cells are made of triplets of either AAB cells, ABA cells, BAA cells, or BBB cells (Set-1).

In both cases, ***f=3*** is odd. The general equations developed in §2-4, eq. (15) to (18) are applicable. When the number of B-cell is even $P_C < P'_C$, and are reversed when the number of B-cell is odd $P_C > P'_C$. The resulting level of non-randomness, $\lambda_C=3.2\ 10^{-5}$, is **25x25=625** times smaller than the level of non-randomness before the XOR compiler, $\lambda_A=2\ 10^{-2}$. It is interesting to notice that a 3-bits XOR data compiler needs only 50% more starting cells than a 2-bits compiler, and is showing a level of non-randomness 25 times lower.

978-1-5090-5443-5/17/$31.00 ©2017 IEEE

III. EXPERIMENTAL ANALYSIS

A. Experimental data

This paper is based on the study of resistive random access memory (ReRAM), an emerging technology that provides a low power alternative to flash. The cells of ReRAMs, see references [38-45], are based on a stack of two electrodes separated by a solid electrolyte. Differential voltages applied on these stacks can move positively charged elements such as oxygen vacancies or silver ions, and short the stack.

Cu/TaOx/Pt resistive crossbar arrays have been fabricated on thermally oxidized Si wafers, Reference [38]. The Cu/TaOx/Pt switches from "0" to "1" based on the formation and the rupture of filaments, made of oxygen vacancies, bridging the dielectric between both electrodes. It exists a minimum Vset voltage applied across the switch, at which a filament is being formed. When the voltage is applied to the stack, the current remains low until Vset is reached, then the current becomes orders of magnitude higher. This effect is reversible, and the filaments can partially be dissolved with opposite voltages applied to the stack.

The parameter $\mathcal{P}$ that is analyzed for the purpose of TRNG is the Vset of the array of these metal oxide resistive cells. The entire population of all cells of the array has a Vset distribution that is well represented by a normal distribution having a standard variation $\sigma_{Array}=0.5V$ and a median value of $2.1V$. The repetitive measurement of the Vset of each cells is also well represented by a normal distribution having a standard distribution $\sigma_{Cell}=0.1V$.

For the purpose of random number generation, the Vset of each cell is measured; a cell is considered as a "0" state when $Vset<2.1V$, and a "1" state when $Vset>2.1V$ of. The cells having average Vset measurements at or close to 2.1 Volt, are good candidates for TRNG. The five populations described below are subsets of the total distribution of cells of the array:

- Case-1: Only ***2%*** of the cells are the ternary states "X" which are used to generate random number.

For these cells parameter $\mathcal{P}$ is very close to the transition of 2.1 Volt. Half of the cells, the A-cells, have $P_A=0.52$ probabilities to be a "1", $P'_A=0.48$ probabilities to be a "0", with $\lambda_A=2\ 10^{-2}$. The second group, the B-cells, have $P_B=0.48$ probabilities to be a "1", $P'_B=0.52$ probabilities to be a "0", with $\lambda_B=\lambda_A=2\ 10^{-2}$.

- Case-2: **4%** of the cells are ternary states. The probabilities as defined above are:

  $P_A=P'_B=0.54$; $P'_A=P_B=0.46$; $\lambda_B=\lambda_A=4\ 10^{-2}$.
- Case-3: **7%** of the cells are ternary states. The probabilities as defined above are:

  $P_A=P'_B=0.56$; $P'_A=P_B=0.44$; $\lambda_B=\lambda_A=6\ 10^{-2}$.
- Case-4: **11%** of the cells are ternary states. The probabilities as defined above are:

  $P_A=P'_B=0.60$; $P'_A=P_B=0.40$; $\lambda_B=\lambda_A=1\ 10^{-1}$.
- Case-5: **100%** of the cells are included. The probabilities as defined above are:

  $P_A=P'_B=0.90$; $P'_A=P_B=0.10$; $\lambda_B=\lambda_A=4\ 10^{-1}$.

In this last case, there are no ternary states, the entire memory array is used to generate random numbers.

The reason we are considering this range of options is to quantify the effectiveness of the XOR data compiler to generate a random number as a function of how tight the ternary state distribution is. Case-1 is the one with the highest initial randomness, while Case-5 is the lowest one.

B. Analysis of the experimental data

The probabilistic model of §II is used to analyze the five experimental cases presented §3-1. Table 1 and Fig. 8 the impact of the XOR data compiler is presented when f varies from 2 to 5.

TABLE I. LEVEL OF NON-RANDOMNESS λ BY EXPERIMENTAL CASE

Case	% of X-cells	λ_A Initial	λ_C 2 XOR	λ_C 3 XOR	λ_C 4 XOR	λ_C 5 XOR
Case-1 52% - 48%	2%	$2\ 10^{-2}$	$8\ 10^{-4}$	$3.2\ 10^{-5}$	$1.28\ 10^{-6}$	$5.12\ 10^{-8}$
Case-2 54% - 46%	4%	$4\ 10^{-2}$	$3.2\ 10^{-3}$	$2.56\ 10^{-4}$	$2.15\ 10^{-5}$	$1.64\ 10^{-6}$
Case-3 56% - 44%	7%	$6\ 10^{-2}$	$7.2\ 10^{-3}$	$8.64\ 10^{-4}$	$1.04\ 10^{-4}$	$1.2\ 10^{-5}$
Case-4 60% - 40%	11%	$1\ 10^{-1}$	$2\ 10^{-2}$	$4\ 10^{-3}$	$8\ 10^{-4}$	$1.6\ 10^{-4}$
Case-5 90% -10%	100%	$4\ 10^{-1}$	$3.2\ 10^{-1}$	$2.38\ 10^{-1}$	$2.05\ 10^{-1}$	$1.64\ 10^{-1}$

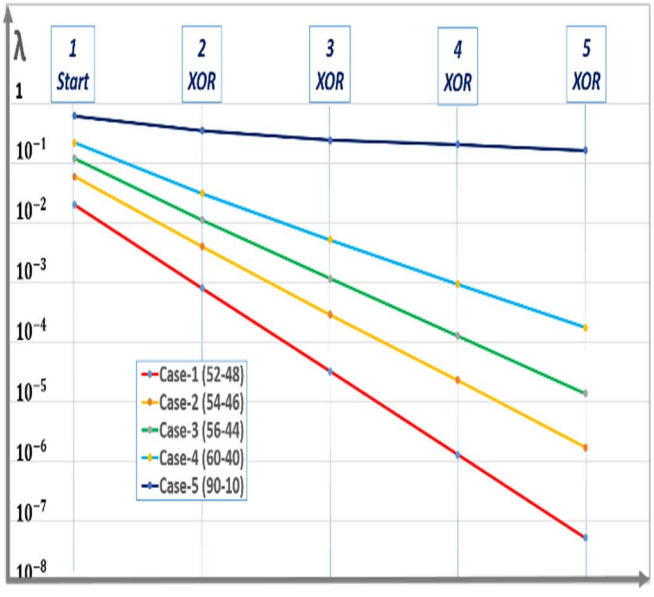

Fig. 8. Efficiency of the XOR data compiler

978-1-5090-5443-5/17/$31.00 ©2017 IEEE

The first observation is the lack of efficiency of the XOR compiler on case-5, the one without ternary states. The lack of initial randomness of this case is such that the method presented in this paper is weak. In all other cases the XOR data compiler when combined with the "X" ternary states is very efficient. Case-1 which has the highest level of initial randomness, has the XOR compiler with the highest efficiency: with 5 XOR, $\lambda_C = 5.12\ 10^{-8}$ which is a very small deviation from absolute randomness.

TABLE II. PREDICTIVE MODEL FOR A GIVEN OBJECTIVE IN λ

Case	% of X-cells	For $\lambda_C < 5\ 10^{-8}$ Round *f* of XOR	For $\lambda_C < 1\ 10^{-10}$ Round *f* of XOR
Case-1 52% - 48%	2%	5	7
Case-2 54% - 46%	4%	7	9
Case-3 56% - 44%	7%	8	12
Case-4 60% - 40%	11%	10	14

It is strait forward to use the model as a predictive tool. For example, as shown Table 2 it is possible to calculate the number ***f*** necessary to get $\lambda_C < 10^{-10}$ for Cases 1 to 4. This confirms that the efficiency of the XOR compiler is higher when combined with higher initial randomness, i.e case-1.

IV. OTHER USES OF THE XOR DATA COMPILER

A. *Effect of physical parameter drifts*

The initial source of randomness of this TRNG is a physical element. This is a fundamental strength compared with mathematically generated pseudo RNG (PRNG) because mathematical algorithms can be discovered by hackers, while unclonable physical elements are random due to micro-variations during manufacturing, and natural noise effect during measurements.

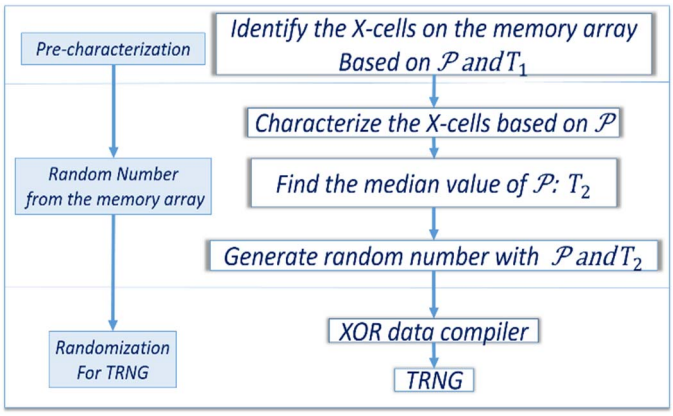

Fig. 9. Flow to reduce the effect of a parameter drift

However physical elements can vary often in a predictable way when subject to effects such as temperature change, biasing conditions, and induced attacks. For example, the value of the Vset of a resistive RAM goes down when subject to higher temperature. A hacker could submit the physical element to a hot air blower to increase temperature, reduce Vset, thereby making both A-cells and B-cells appear similar, with a high probability to be tested as "0". This could result in much lower randomness with most cells at "0".

The remedy of such an attack is to make the size of the population of A-cells and B-cells equivalent, by adjusting the threshold (T2) between the "0" states, and "1" states, in the median point of the X-cell distribution, see Fig 9:

- Step-1: Sorting out of the instable cells, the X-cells with high levels of randomness.

The sorting is based on the value of parameter $\mathcal{P}$, and a threshold T_1 which differentiates the zeros from the ones, while keeping track of the mapping of the "x" cells.

- Step-2: Prior to the random number generation, all X-cells are retested with parameter $\mathcal{P}$.
- Step-3: Identification of the threshold T_2 placed at the median of the population. By design half of the X-cells are below T_2, and half are above T_2.
- Step-4: Program the X-cells, "0" below T_2, and "1" above T_2.
- Step-5: Extract the corresponding data stream, and use the XOR compiler to enhance the randomness of the data stream to generate the TRNG.

With this method the raw data stream generated by the memory array and the X-cell has a population with equal numbers of A-cells and B-cells, and this in spite of a potential drift in temperature caused by the hot air blower of the hacker. The integrity of the TRNG is thereby protected. The parameter $\mathcal{P}$ can drift when subjected to many effects such as voltage and current bias, noise, and electromagnetic interferences. The method described Fig. 9.0 should be effective for these various drifts of parameter $\mathcal{P}$.

B. *Generalization of the TRNG design*

In this paper it is suggested that the TRNG can be generated by a PUF based memory array that can concurrently generate CRPs. The method is applicable to any memory device as long as it is possible to identify a parameter $\mathcal{P}$ that can be tested to sort out the cells, and identify enough unstable X-cells. The algorithm presented Fig. 9.0 is generic, and applicable to the following examples:

- Flash or EEPROM memory: parameter $\mathcal{P}$ could be the time to charge the floating gate to get a particular threshold voltage of each flash cell, or the trans-conductance of the cell after programming.
- DRAM memory: parameter $\mathcal{P}$ could be the measurement of the residual charge left in a cell after constant discharging time.
- ReRAM memory: In addition of the Vset as presented in this paper, parameter $\mathcal{P}$ could be the Vreset (threshold voltage to erase the cells), Roff (resistivity on

978-1-5090-5443-5/17/$31.00 ©2017 IEEE

the high resistance state), or Ron (resistivity on the low resistivity state).

- SRAM memory: TRNG based on the X-cells that are not consistently switching as a clear 0 or a clear 1.

As presented in Fig 10 the method can be generalized to any data stream based on random physical component, and a parameter $\mathcal{P}$.

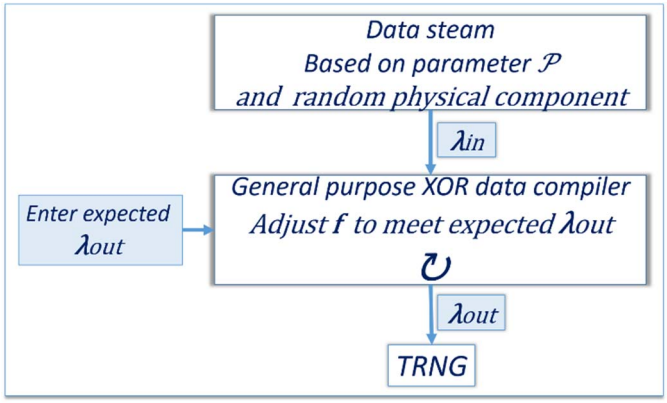

Fig. 10. Generalization of the concept to any data stream

If the physical component generates a data stream with a deviation from absolute randomness λ_{in}, it is possible to model the size f of XOR, as described §II, to meet a particular λ_{out} objective.

V. Planned Future Work

Improved statistical model

The experimental measurements of the ReRAM samples presented in this paper produced a distribution of Vset that can be appropriately described by a normal distribution. The model developed assumes that the initial random number generated from the X-cells can be separated with symmetrically distributed A-cells and B-cells. We are currently studying other statistical distributions beside the normal one, together with more sophisticated segmentation of the X-cells. We are not anticipating that these improved statistical models will significantly deviate from the one we presented in this paper with tight X-cells distribution, i.e. when only X-cell close to the median distribution are selected, however the deviations could be important for wider distribution of the X-cells away from the median.

Hardware implementation with a PUF

We are developing a prototype to study various ReRAM arrays with a state machine designed with field programmable gate arrays (FPGA) that will automatically extract large quantities of PUF CRPs and TRNG from the memory arrays. The XOR data compiler is included in the design of the FPGA. We intend to use the prototype to further validate our statistical models, and to leverage the tools developed by NIST that are available online to quantify the entropy, and the level of randomness of the TRNGs. The prototype should have the built-in flexibility to allow us to analyze multiple types of memory arrays, with different methods of fabrication. We are interested to optimize the properties of the TRNG while reducing CRP error rates, and developing cryptographic protocols that leverage the randomness of PUFs.

VI. Summary and Conclusion

The value of combining a XOR data compiler with a memory based PUF using ternary states to generate true random numbers has been confirmed. We noticed that the XOR data compiler is not effective when the initial data stream is not random.

Based on a normal distribution of parameter $\mathcal{P}$, the proposed statistical model can quantify the deviation from pure randomness of the TRNGs. It is possible to calculate f, the length of the XOR, to reach a desired level of non-randomness λ_C, as a function of the level of non-randomness λ_A of the incoming data stream extracted from the physical element.

An algorithm was suggested to handle the drift of parameter $\mathcal{P}$, when subject to effects such as the ones due to temperature changes. As a result, it is expected that parameter drifts should not degrade the quality of the TRNGs.

The proposed XOR data compiler is strait forward to implement both in software, and in hardware, with a high level of parallelism that can generate a random number in a few clock cycles. In order to further accelerate the process to generate fresh random numbers on demand, the X-cells can be tested in advance, and the data can be stored in the memory. The read time of a ReRAM is typically 10ns/bit, so we believe that the generation of the TRNG has the potential to be done at a rate of 100Mbit/s. Characterizing the data rate of the TRNG will be part of the planned future work described §V.

One of the key objective of this study has been to develop cryptographic primitives to secure the Internet of Things (IoT). The implementation of affordable cryptographic protocols securing IoTs can benefit from the ease of use of PUFs which can reduce the complexity of key distribution. TRNGs are essentials for the IoTs to encrypt challenges, responses, and other cryptographic keys. We are convinced that the solution of a XOR data compiler leveraging mainstream embedded memories of IoTs is practical, and only requires light resources to be implemented.

The method described in this paper can also be used in the study of swarm dynamics by generating true random noises [46], and other similar applications requiring TRNG.

Aknowledgment

Dr. Marius Orlowski from Virginia Tech produced, and characterized the metal oxide ReRAM samples that were presented at CISR 2016 [29]. These measurements were used to prepare the experimental analysis of § III. Dr. Derek Sonderegger from the Department of Mathematics and Statistic of Northern Arizona University provided some valuable input to the development of a statistical model § 2.4. The author is thanking both Dr. Orlowski, and Dr. Sonderegger for their support in this work.

References

[1] M. Stipevic, and all; true random number generator; Open problems in Mathematics and computational science; *pp.275-315, Springer, 2014;*

978-1-5090-5443-5/17/$31.00 ©2017 IEEE

[2] A. Maiti, and all; PUF and TRNG: a compact and scalable implementation; *GLSVLSI'09, May 2019, Boston;*

[3] D. Glosemeyer, and B Knapp; Random Number Generation; Wolfram Mathematical Tutorial collection, 2008;

[4] H. Katzgraber; Random Numbers in scientific computing: an introduction; *International summer school modern computational science, aug 2010, Oldenburg, Germany*

[5] R. Soorat; Hardware random number generation for cryptography; https://arxiv.org/pdf/1510.01234 , 2015*;*

[6] Berndt Gammel, and all; Jul 2012; Random Generator configured to combine states of memory cells; *US patent No 7,979,482 B2;*

[7] Sang-Yong Yoon, and all; Aug 2012; Method of operating nonvolatile memory devices storing randomized data generated by copyback operation; *US patent No: 8990481 B2;*

[8] Ravikanth Pappu, Ben. Recht, Jason Taylor, and Neil Gershenfield; 20 Sept 2002; Physical one-way functions; *Science. Vol 297 No5589 pp2026-2030.*

[9] R. Maes; Physically Unclonable functions: constructions, properties, and applications; *Doctoral thesis- Catholic University of Leuven, 2012;*

[10] Yier Jin; Introduction to hardware security*, Electronics 2015, 4, 763-784; doi:10.3390/electronics4040763.*

[11] Herder, C., and all; "PUFs and Applications; *A Tutorial." Proceedings of the IEEE 102, no. 8 (2014): 1126-1141;*

[12] B. Gassend, and all; Silicon PUFs; *CCS' 2002;*

[13] B. Gassend; Physical random functions; *M.S. thesis, Dept. Electr. Eng. Comput. Sci., MA, USA, Massachusetts Inst. Technol., Cambridge, 2003;*

[14] S. Katzenbeisser, and all; PUFs: myths, fact or busted? A security evaluation of PUFs cast in silicon; *CHES 2012;*

[15] David. Naccache and Patrice. Frémanteau; Aug. 1992; Unforgeable identification device, identification device reader and method of identification; *Patent US5434917.*

[16] Y. Gao, and all; Emerging Physical Unclonable Functions with nanotechnologies; *IEEE, DOI: 10.1109/ACCESS.2015.2503432*;

[17] Guajardo, J, and all; PUFs and PublicKey Crypto for FPGA IP Protection; *Field Programmable Logic and Applications, 2007.*

[18] Guajardo, J., and all; FPGA Intrinsic PUFs and Their Use for IP Protection; *CHES, 2007;*

[19] Maes, and all; A Soft Decision Helper Data Algorithm for SRAM PUFs; *2016 IEEE International Symposium on Information Theory. 2009;*

[20] Daniel E. Holcomb, Wayne P. Burleson, Kevin Fu; Nov 2008; Power-up SRAM state as an Identifying Fingerprint and Source of TRN; *IEEE Trans. on Comp., vol 57, No 11.*

[21] R. Maes, P. Tuyls and I. Verbauwhede, "A Soft Decision Helper Data Algorithm for SRAM PUFs," in *2009 IEEE International Symposium on Information Theory*, 2009.

[22] Daniel E Holcomb, and all; Power up SRAM state as an identifying Fingerprint and Source of True Random Numbers; *IEEE Trans. On Computers, 2009 Vol 58, issue No09 Sept;*

[23] Pravin Prabhu, Ameen Akel, Laura M. Grupp, Wing-Kei S. Yu, G. Edward Suh, Edwin Kan, and Steven Swanson; June 2011; Extracting Device Fingerprints from Flash Memory by Exploiting Physical Variations; *4th international conference on Trust and trustworthy computing*;

[24] Todd A. Christensen, John E Sheets II; Oct. 30, 2012; Implementing PUF utilizing EDRAM memory cell capacitance variation*; Patent No.: US 8,300,450 B2.*

[25] Xiaochun Zhu, Steven Millendorf, Xu Guo, David M. Jacobson, Kangho Lee, Seung H. Kang, Matthew M. Nowak, Daha Fazla; March 2015; PUFs based on resistivity of MRAM magnetic tunnel junctions; *Patents. US 2015/0071432 A1.*

[26] Elena I. Vatajelu, Giorgio Di Natale, Mario Barbareschi, Lionel Torres, Marco Indaco, and Paolo Prinetto; July 2015; STT-MRAM-Based PUF Architecture exploiting Magnetic Tunnel Junction Fabrication-Induced Variability; *ACM transactions.*

[27] An Chen; 2015; Comprehensive Assessment of RRAM-based PUF for Hardware Security Applications; *978-1-4673-9894-7/15/IEDM IEEE.*

[28] Christian Krutzik; Jan 2015; Solid state drive Physical Unclonable Function erase verification device and method; *US Patent Application publication US 2015/0007337 A1.*

[29] B. Cambou, and M. Orlowski; PUF designed with ReRAM and ternary states; *CISR 2016, April 2016, Oak ridge;*

[30] Dai Yamamoto, Kazuo Sakiyama, Kazuo Ohto, and Masahiko Itoh; 2011; Uniqueness Enhancement of PUF Responses Based on the Locations of Random Outputting RS Latches; *CHES 2011 Lecture Notes in Computer Science Volume 6917, pp 390-406.*

[31] Anuj Gupta, May 2005, Implementing Generic BIST for testing Kilo-Bit Memories; *Master Thesis No-6030402 Deemed University Patiala India.*

[32] J. Soto; Statistical testing of random number generators; 1999 – NIST; http://csrc.nist.gov/rng/rng5.html ;

[33] P. L'Ecuyer; Software for uniform random number generation: distinguishing the good from the bad; *2001 Simulation Conference*;

[34] A. Rukhin, and all; A statistical test suite for random and PRNG for cryptographic applications; *April 2010, special publication from NIST 800-22 rev 1a;*

[35] Marcos C Tzannes, Arnon Friedmann; Nov 2001; Randomization using an XOR scrambler in multicarrier communications; *US patent No 9191939 B2*;

[36] R. Davies; XOR and hardware random number generator; http://www.robertnz.net/pdf/XOR2.pdf; *February 2002;*

[37] George Marsaglia; XOR shift RNG; *Journal of Statistical software, 2003.*

[38] Gargi Ghosh and Marius Orlowski; 2015; Write and Erase Threshold Voltage Interdependence in Resistive Switching Memory Cells; *IEEE trans. on Electron Devices, 62(9), pp. 2850-2857.*

[39] Gilbert, Nad, et al. "A 0.6 V 8 pJ/write Non-Volatile CBRAM Macro Embedded in a Body Sensor Node for Ultra Low Energy Applications." *VLSI Circuits (VLSIC), 2013 Symposium on. IEEE, 2013*;

[40] M. N. Kozicki, M. Park, and M. Mitkova, "Nanoscale memory elements based on solid-state electrolytes," *IEEE Trans. Nanotechnol, vol. 4, pp. 331-338, May 2005;*

[41] M. N. Kozicki and M. Mitkova, "Mass transport in chalcogenide electrolyte films – materials and applications," *J. of Non-Crystalline Solids, vol. 352, pp. 567-577, March 2006;*

[42] Valov, R. Waser, J. R. Jameson and M. N. Kozicki, "Electrochemical metallization memories - Fundamentals, applications, prospects," *Nanotechnology, vol. 22, p. 254003, June 2011;*

[43] M. N. Kozicki, C. Gopalan, M. Balakrishnan, M. Park and M. Mitkova, "Nonvolatile memory based on solid electrolytes," in *Proc. IEEE Non-Volatile Memory Technol. Symp.*, 2004;

[44] Patrick R. Mickel, Andrew J. Lohn, Byung Joon Choi, J. Joshua Yang, Min-Xian Zhang, Matthew J. Marinella, Conrad D. James, and R. Stanley Williams, "A physical model of switching dynamics in tantalum oxide memristive devices," *Appl. Phys. Lett., vol. 102, p. 223502, 2013*;

[45] M. N. Kozicki, M. Balakrishnan, C. Gopalan, C. Ratnakumar, and M. Mitkova, "Programmable metallization cell memory based on Ag-Ge-S and Cu-Ge-S solid electrolytes," *Proc. NVMTS, p. 8389, 2005;*

[46] Y. Shang, and R. Bouffanais; "Influence of the number of topologically interacting neighbors on swarm dynamics"; Scientific Reports DOI: 10.1038/srep04184, 2015.

978-1-5090-5443-5/17/$31.00 ©2017 IEEE