Towards a Memristive Hardware Secure Hash Function (MemHash)

Leonid Azriel and Shahar Kvatinsky
Viterbi Faculty of Electrical Engineering
Technion – Israel Institute of Technology, Haifa, Israel
Email: leonida@tx.technion.ac.il, shahar@ee.technion.ac.il

***Abstract*—Hardware based hash functions might provide a low cost and low power alternative to the classic solutions, which are based on implementations of mathematical cryptographic algorithms. In this paper, we propose MemHash, a hardware secure hash function built using memristive technology that exploits the unique properties of memristors. The MemHash operation is based on intrinsic device characteristics. Furthermore, it exploits process variations for implicit key embedding, thus creating a keyed-hash message authentication code (HMAC) that does not involve a separate key generation and management process. MemHash comprises a memristive crossbar with a differential read mechanism and a scrambler unit. The scrambler unit receives the input message as a bit stream and digitally mixes it with data read from the array. For every bit of the message, the scrambler generates a write address and a value to perform a single-cell write cycle to the crossbar. Because the crossbar is designed to be extremely sensitive to the write disturb phenomenon, every single-cell write alters additional cells in the design, thus increasing the entropy. The differential read mechanism provides sensitivity to process variations and robustness in operating conditions, yielding a PUF-like effect. MemHash is evaluated with a 16×16 memristive crossbar structure. Our simulation results demonstrate the statistical characteristics of the proposed design, showing close-to-optimal uniqueness and diffuseness.**

I. Introduction

Emerging nanoelectronic memristive technologies, such as Resistive RAM (ReRAM), Phase Change Memory (PCM), and Spin-Torque Transfer Magnetoresistive RAM (STT-MRAM), promise to provide an alternative to the ubiquitous CMOS technology, aiming to replace the conventional DRAM and Flash memories. No less important, the unique properties of memristors and their compatibility with the CMOS process make it possible to create conceptually new digital and analog circuits. CMOS compatibility, along with a combination of properties such as non-linearity, non-volatility and sensitivity to process variations, make this technology appealing for embedded security applications.

The memristor is a passive device that changes its resistance under an applied electrical current. Chua coined the term memristor in his seminal work [1], where he speculated on the existence of a fourth passive element. Following the publication by Strukov *et al.* in 2008 [2], where Chua's memristor theory was linked to the resistive switching phenomenon, the interest in memristor research, including security with memristors, has grown significantly. Several research groups reported on building security primitives with memristor technology. For example, true random number generators have been proposed, leveraging the stochastic properties of memristors [3] and telegraph noise in resistive RAM [4]. Another important security primitive is the Physical Unclonable Function (*PUF*), which utilizes intrinsic process variations to generate a unique response to a challenge per device instance. Recently, several researchers suggested exploiting the unique characteristics of memristive devices to increase the entropy and sensitivity to process variations in PUFs [5], [6]. Conceptually new memristive PUF architectures have also been proposed. Chen *et al.* [7] presented a fabricated PUF device that comprises a memristive crossbar memory array, where process-variation based differences between memristor resistance values serve as an entropy source. Rose *et al.* [8] investigated a memristive crossbar PUF that utilized variations in memristor write time as well as the parasitic sneak path effect during read.

In the spirit of the PUF concept, we propose *MemHash*, a memristive hardware hash function. MemHash employs a memristive crossbar array that exploits the parasitic write disturb phenomenon to achieve higher entropy, and it takes advantage of process variations to create a unique and unclonable key per instance. However, MemHash serves different purposes than PUF. Unlike PUF, MemHash accepts messages with arbitrarily long length. Thus, it is essentially a keyed secure compression function. In MemHash, the key is embedded in the circuit and cannot be exported. Thus, MemHash suits applications where the generator and the verifier of the signature are the same entity. For example, it can be used for integrity check of a local memory.

The secure hash function is a fundamental component of modern cryptography. It enables important security applications such as digital signature and integrity validation. The state-of-the-art secure hash functions are based on mathematical algorithms, such as SHA, SHA-2, SHA-3 or MD5. Pure software implementations of these algorithms often fail to provide sufficient performance and security. Hence, hardware accelerators that use digital logic to implement the hash algorithms are commonly used. To construct an HMAC, the key is managed separately. In contrast to the classic approach, pure hardware implementations of secure hash functions utilize intrinsic properties of the hardware to create entropy. Few pure hardware hash implementations have been proposed, and those have been mainly based on chaotic systems [9], [10]. Intrinsic implementations of secure hash functions promise great improvement in power and area efficiency. However,

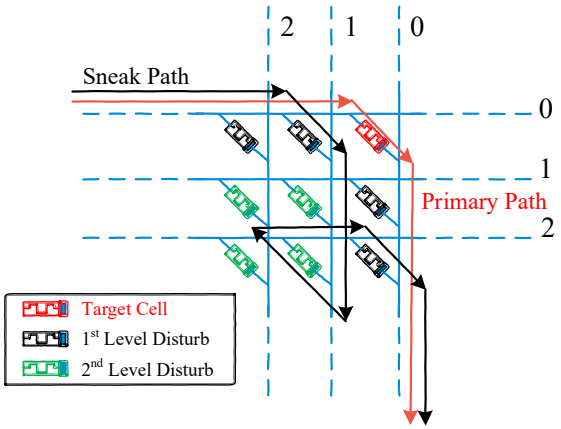

Fig. 1. Memristive crossbar memory segment during a write. The target cell is the cell selected for the write. The cells sharing the same row or column of the selected cell are the first-level disturb cells. They change to the same direction as the target cells. The remaining cells are the second-level disturb cells. Typically, they change to the opposite direction, and they get the least current.

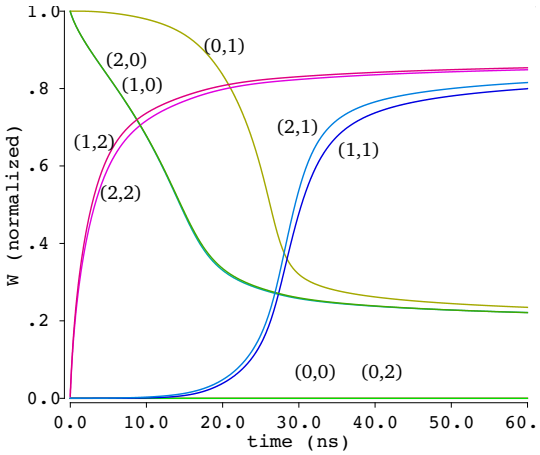

Fig. 2. Single write operation to a 3×3 memristive crossbar array. The vertical axis shows a normalized internal state of the memristors (W). $W = 0$ means Low Resistance State, and $W = 1$ means High Resistance State.

they fail to provide sufficient robustness. MemHash uses discrete quasistable states, in which the memristor drift rates are sufficiently slow, along with differential reads to increase robustness and accuracy.

II. MemHash Design

A. Crossbar Array and Write Disturb

The crossbar is a fundamental structure used for building memory with resistive cells. It requires no additional elements other than the memristors, which makes it superior in area efficiency. However, a significant drawback of the crossbar is in the sneak paths, parasitic current paths through unselected memory cells, resulting from the absence of switching elements in the array [11]. The sneak paths distort the read value during read and modify unselected cells during write (write disturb). These side effects make it difficult to build large and robust crossbar memory arrays. However, side effects that add entropy, if made predictable, can be harnessed for security applications [12]. One such application is a secure hash function.

To create a secure hash function, we consider the state of the entire array as a hash state. Individual cells are selected for writing based on the input message and the previous state. No write disturb mitigation is implemented. Hence, during a write, in addition to the target cell, other cells are modified. We define two types of cells in this respect (Figure 1). First-level disturb cells are the cells that share either the row or the column with the target cell. The resistances of these cells change to the same direction as the target cell. All the remaining cells are second-level disturb cells, and, typically, their resistance changes to the direction opposite to the target cell. The number of first-level disturb cells in an n-bit array is $O(\sqrt{n})$, and the number of second-level disturb cells is $O(n)$. Hence, for current-controlled memristors, the latter get less current and, as a result, change more slowly.

Every write operation places the array in a new state that depends on the previous state, the write address and the write data. The number of states must be sufficiently large to resist brute-force and modeling attacks. Binary encoding — i.e., using a high resistance state (*HRS*) and low resistance state (*LRS*) — limits the state space, since, due to the write disturb phenomenon, all the disturb cells of the same level move towards the same value. To increase the state space, the MemHash design allows intermediate resistance states for the memristor cells, in addition to the full low and high resistance states. This is achieved by using a voltage level sufficiently low so that the system will reach equilibrium, where all the memristor resistances remain at intermediate values as a result of the current having been reduced below the threshold. We define this state as *stable*. However, reaching the stable state may take an unacceptably long time. Additionally, the number of possible stable states may be insufficient from a security perspective, although that is still an open question. We define as *quasistable* a state in which the rate of change of all the cells in the array is lower than a certain threshold, so that the state may be considered as stable for practical purposes.

To demonstrate a single step, consider a 3×3 crossbar array in a similar structure as shown in Figure 1. Cell (i, j) connects row i with column j. Assume the initial state of the array is as follows: cells $(0,1)$, $(1,0)$ and $(2,0)$ are in *HRS*, and all the remaining cells are in *LRS*. A negative voltage is applied between row 0 and column 0. When applied to a bipolar memristor, the negative voltage changes the state of the memristor toward *LRS*. Figure 2 illustrates the response of the memristors' internal states to the write pulse. The target cell $(0,0)$ is already in *LRS*, hence the cell remains unaltered. Cell $(0,2)$ is also at low resistance; hence, relatively high positive current flows through cells $(1,2)$ and $(2,2)$, causing them to start moving towards *HRS*. In parallel, column 0's first-level disturb cells move towards *LRS*. Cell $(0,1)$ starts moving in

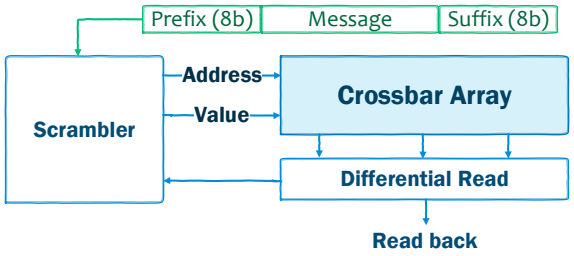

Fig. 3. MemHash Architecture. The original message is wrapped with a constant prefix and a suffix. The resulting message is fed serially to the scrambler. The scrambler implements a linear function of the input bit, the cycle count and a value read from the array, from which it generates an address and a value for writing into the array. The array is connected to a differential read circuit that produces an input to the scrambler for the next cycle and a signature read-back.

the same direction, but with a delay. The delay results from smaller current flowing through this cell, due to the parallel connection with $(0,2)$. Once the resistance of $(0,1)$ is reduced, the neighboring second-level disturb cells $(1,1)$ and $(2,1)$ get sufficient current to change their states. At the end, the system is stabilized at a new quasistable state. The slight differences between the change rates of different cells, e.g., cells $(1,1)$ and $(2,1)$, result from variations in their electrical parameters. This way, the process variations are simulated.

Memristive devices are generally sensitive to process variations [13]. PUF designs exploit this sensitivity to create unique and unclonable instances on each unit. This is also a crucial component of the MemHash concept. Process variations serve two goals: (1) They increase entropy and generate a complex circuit resistant to modeling attacks, and (2) they generate a keyed hash function with a unique and unclonable key per instance. While sensitivity to the process is essential for the MemHash design, the circuit must provide reliable results under different operating conditions. MemHash addresses this with a differential read [14]. The columns are divided to pairs, and each pair is connected to the two inputs of a sense amplifier. Thus, unlike a conventional single-ended read mode, differential read essentially compares the resistances of two adjacent cells in the addressed row. Besides providing reliability, the differential read also obfuscates the array contents from the user. Rather than reading the memristor values, an attacker can only see the results of a comparison, which complicates modeling attacks. In contrast to reads, writes are always single ended.

The size of the memory array must be sufficiently large to prevent birthday attacks that find collisions within the time of $O(\sqrt{2^k})$, where k is the number of bits in the hash value. This means that the hash value must be at least 128 bits. Due to the differential read, the size of the crossbar array must be twice as big, that is 256 (16×16) bits.

B. MemHash Architecture

Figure 3 illustrates an 8-bit MemHash architecture. The user message is first wrapped with an 8-bit constant prefix and suffix words. MemHash processes the input stream serially, one bit per cycle. The scrambler maintains an internal wrap-around counter counting modulo the number of rows. Every cycle the scrambler reads from the row pointed by the counter. The value read from the row is mixed with the current bit to compose the address (row and column) of the target cell. The value to be written (0 or 1) is determined by performing an exclusive OR between all the bits of the row and the current bit of the message.

To guarantee repeatability, the array is initialized to a known state at the beginning of the hashing process. In the initial state, each cell is in either the full *LRS* or the full *HRS*. The initialization write must be accurate: namely, write disturb must be prevented. This can be achieved, for example, by a slow write with lower voltage. The 8-bit prefix brings each instance of the array to a different random state, and the 8-bit suffix prevents attacks that modify the trailing bits in the message. The detailed algorithm and the scrambling function of the MemHash Scrambler block is provided below (Algorithm 1).

Algorithm 1 MemHash Scrambler Algorithm

Initialize array to alternating LRS/HRS values
$pointer = 0$
$prefix = 10101010$
$suffix = 01010101$
$input = \{prefix, message, suffix\}$
while *input* not empty **do**
  $b = pop\ 1\ bit from\ input$
  $row =$ (4 LSBs from $Array[pointer] + b$) % $numrows$
  $col =$ (4 MSBs from $Array[pointer] + b$) % $numcols$
  $writevalue = XOR(row\%2, col\%2, b)$
  Write $writevalue$ to cell address $\{row, col\}$
  $pointer = (pointer + 1)$ % $numrows$
end while

C. Memristor Model and Parameters

Memristors can be either voltage or current controlled. The regular structure of the crossbar makes the voltage controlled model more predictable; hence we chose the current controlled model for the hash application. For simulations, we used the TEAM model [15]. We simulate the process variations by modifying five parameters of the model. R_{on}, R_{off} and D reflect variance in resistance, and i_{on} and i_{off} reflect variance in the change rate.

We selected the memristor device parameters to maintain most of the memory cells at intermediate resistance for any given balanced sequence of operations. The scrambling function guarantees the balance of '0' and '1' write values.

III. Experimental Results

A. Experimental setup

For performance evaluation of MemHash, we simulated the write operations in a 16×16 memristive crossbar using SPICE. The scrambler, the differential read module and the message

scheduler were simulated with the Perl script framework. The differential read was emulated by comparing the resistances of the adjacent cells.

TABLE I
MEMRISTOR TEAM MODEL PARAMETERS

Parameter	Value	Standard deviation
Set Voltage	$-0.15V$	0
Reset Voltage	$2.4V$	0
Period	$60ns$	0
D	$3 \cdot 10^{-9}m$	5%
R_{ON}	100Ω	5%
R_{OFF}	$20K\Omega$	5%
i_{on}	$-1\mu A$	5%
i_{off}	$1\mu A$	5%
α_{on}	3	0
α_{off}	3	0
Window function	Kvatinsky window	N/A

The Perl script prepared a separate SPICE deck for every message bit, translated to a single cycle in MemHash. At the end of each simulation step, the memristor state vector passes to the succeeding step. We used Monte-Carlo to emulate process variations, using the parameters in Table I. The TEAM model parameters were tuned to keep the crossbar array in a balanced state to prevent saturation.

B. Dynamic Behavior

First, we checked the dynamic behavior of the system in a transient simulation. For this purpose, we simulated the MemHash system with 10 randomly generated 32-bit message pairs. In each pair, the two messages differ by a single bit (Hamming Distance = 1) located in the middle of the message (bit 15). After each cycle, we measure the Hamming distance between the MemHash differential states of the two messages. The hash size is 128; therefore, we expect a Hamming distance of 64 at maximum entropy. Figure 4 shows the Hamming distances for the message pairs, starting from the differentiating bit. For nine of them, the Hamming distance grows to around 40% within eight cycles after the appearance of the differentiating bit. An additional observation is that once the hash values for the messages are separated, the Hamming distance stays in the same zone; namely, no reconvergence is observed. These results indicate that the 8-bit suffix is in most cases sufficient to achieve the avalanche effect. However, for one message pair, the distance grows only after 16 cycles. Note that although the distance between the messages for this pair reaches 0 at cycle 27, the messages end up separated since every next step depends not only on a representation of the array state as a differential binary vector, but also on the analog state of the array.

C. Statistical Properties

Statistical performance of cryptographic hash functions can be evaluated using the confusion and diffusion criteria introduced by Shannon in 1945. Confusion reflects the sensitivity of the hash value to different parts of the key, and diffusion reflects the sensitivity to changes in the input data. Generally, a change in a single bit of the input message must result in a change of 50% of the hash value bits on average. MemHash shares several properties with the PUF structure. Hence, we combine these parameters with the PUF-related criteria defined in [16].

The first parameter, *uniqueness*, reflects the variance between responses to the same message from different instances of the evaluated circuit. Uniqueness is measured in Hamming distance (HD) between the hash values.

The next parameter, $diffuseness$, reflects the variance between hash values for the same instance and different input messages. The *uniqueness* parameter reflects sensitivity to the key, and $diffuseness$ reflects sensitivity to the data, similarly to Shannon's diffusion and confusion respectively. Therefore, we expect both of the parameters to have an average value of 50%. The $uniformity$ parameter reflects the balance between

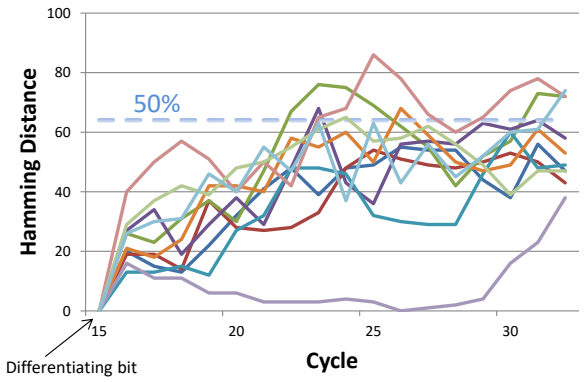

Fig. 4. Dynamic behavior of MemHash. Hamming Distance between hash values of two messages differing in a single bit located in the middle of the message. Every line in the chart represents one such pair. The dotted line is the target distance of 50% of the bits in the hash value.

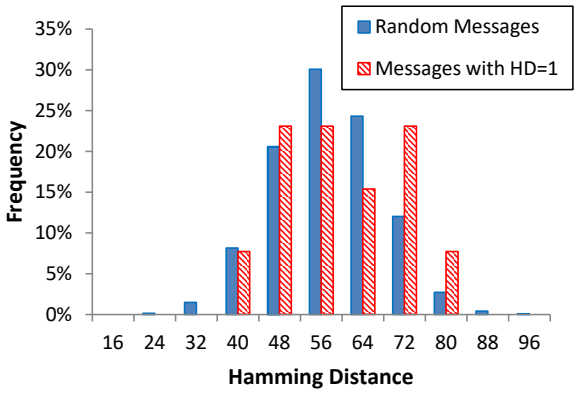

Fig. 5. Diffuseness histogram. Hamming distance between hash values for different messages with the same circuit. The solid bars show the results for random messages, and the striped bars show the results for message pairs that differ in one bit.

0's and 1's in the hash value. It is obtained by summing up the bits in the hash value and dividing the result by the number of bits in it. Finally, the $bit-aliasing$ parameter reflects the balance of 0's and 1's obtained from different instances for every bit individually.

To obtain the diffuseness for MemHash, we generated 32 circuits according to Table I. For every circuit, we ran 32 different random messages. For other parameters, we generated 32 random messages and applied each of them to 32 different circuits. Table II shows the average and standard deviation figures obtained from the simulations. Figure 5 illustrates the diffuseness parameter distribution. For all the parameters, we look to achieve an average of 50% and a small standard deviation. The uniqueness value is almost satisfactory. However, average diffuseness is off by 8% from the target. This can be explained by observing the bit-aliasing numbers for individual bits of the hash values, which reach values of up to 90%. We hypothesize that this phenomenon is caused by certain cell pairs being biased towards a specific differential state. An additional interesting observation is that the diffuseness statistics are identical for random messages and almost identical messages (HD=1). These results suggest that MemHash achieves entropy slightly lower than the maximum possible entropy with the given hash size.

TABLE II
STATISTICAL EVALUATION

Parameter	Average	Standard deviation
Uniqueness	46%	7.5%
Diffuseness (random)	42%	8%
Diffuseness (HD=1)	43%	8%
Uniformity	50%	18%
Bit-aliasing	50%	18%

IV. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a conceptually new design of a keyed hash function, a memristive hardware hash function (MemHash) that exploits the non-linear behavior of memristive devices. MemHash takes advantage of the write disturb phenomenon in memristive crossbar arrays, when writing to one cell affects other cells in the array. In addition, we exploited manufacturing process variations to create unique keys for each instance of the function.

The MemHash concept calls for further development of analytical models for comprehensive security analysis. For example, MemHash can be viewed as a dynamical system, in which time is a discrete space represented by cycles, and state space is the collection of all possible quasistable states of MemHash.

In security analysis, various cryptanalysis techniques should be applied to MemHash to evaluate the one-wayness and collision-free properties. The obtained diffuseness and uniqueness values show that MemHash generates entropy slightly lower than the maximum. This weakness can be exploited to find collisions. In addition, modeling attacks try to extract the parameters of the system (in our case, the electrical parameters of the memristors) by applying different inputs and solving a system of equations. We believe that the differential read scheme makes modeling attacks on MemHash unlikely.

Finally, the reliability (robustness) of MemHash, as well as the accuracy requirements for analog components such as the differential read block, should be further investigated.

ACKNOWLEDGMENT

This research was partially supported by Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI), by the Viterbi Fellowship in the Technion Computer Engineering Center, and by the Hasso Plattner Institute (HPI).

REFERENCES

[1] L. O. Chua, "Memristor—The Missing Circuit Element," *Circuit Theory, IEEE Transactions on*, vol. 18, no. 5, pp. 507–519, 1971.

[2] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The Missing Memristor Found," *Nature*, vol. 453, no. 7191, pp. 80–83, May 2008.

[3] Y. Wang, W. Wen, H. Li, and M. Hu, "A Novel True Random Number Generator Design Leveraging Emerging Memristor Technology," in *Proc. of the 25th Ed. of the Great Lakes Symp. on VLSI - GLSVLSI '15*. New York, NY, USA: ACM Press, May 2015, pp. 271–276.

[4] C.-Y. Huang, W. C. Shen, Y.-H. Tseng, Y.-C. King, and C.-J. Lin, "A Contact-Resistive Random-Access-Memory-Based True Random Number Generator," *IEEE Electron Device Letters*, vol. 33, no. 8, pp. 1108–1110, Aug 2012.

[5] J. Rajendran, R. Karri, J. Wendt, and B. Wysocki, "Nanoelectronic Solutions for Hardware Security," in *Asian South Pacific Design Automation Conference*, 2013, pp. 1–9.

[6] J. Mathew, R. S. Chakraborty, D. P. Sahoo, Y. Yang, and D. K. Pradhan, "A Novel Memristor-Based Hardware Security Primitive," *ACM Transactions on Embedded Computing Systems*, vol. 14, no. 3, pp. 1–20, Apr 2015.

[7] P.-Y. Chen, R. Fang, R. Liu, C. Chakrabarti, Y. Cao, and S. Yu, "Exploiting Resistive Cross-point Array for Compact Design of Physical Unclonable Function," in *2015 IEEE Int. Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, May 2015, pp. 26–31.

[8] G. S. Rose and C. A. Meade, "Performance Analysis of a Memristive Crossbar PUF Design," in *Proceedings of the 52nd Annual Design Automation Conference - DAC '15*. New York, New York, USA: ACM Press, 2015, pp. 1–6.

[9] K. J. D. A. Virtudez and R. C. Gustilo, "FPGA Implementation of a One-way Hash function Utilizing HL11-1111 Nonlinear Digital to Analog Converter," in *TENCON 2012 IEEE Region 10 Conference*. IEEE, Nov 2012, pp. 1–5.

[10] A. E. Edang, J. K. O. Leynes, R. L. A. Ella, R. C. Labayane, and C. M. Santiago, "Nonlinear Maps from Closed-loop Tandems of A-to-D and D-to-A Converters," *Communications in Nonlinear Science and Numerical Simulation*, vol. 16, no. 3, pp. 1483–1489, 2011.

[11] M. A. Zidan, H. A. H. Fahmy, M. M. Hussain, and K. N. Salama, "Memristor-Based Memory: The Sneak Paths Problem and Solutions," *Microelectronics Journal*, vol. 44, no. 2, pp. 176–183, 2013.

[12] S. Kannan, N. Karimi, O. Sinanoglu, and R. Karri, "Security Vulnerabilities of Emerging Nonvolatile Main Memories and Countermeasures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 1, pp. 2–15, Jan 2015.

[13] D. Niu, Y. Chen, C. Xu, and Y. Xie, "Impact of Process Variations on Emerging Memristor," in *Proc. of the 47th Design Automation Conf. - DAC '10*. New York, NY, USA: ACM Press, 2010, p. 877.

[14] L. Zhang, X. Fong, C.-H. Chang, Z. H. Kong, and K. Roy, "Feasibility Study of Emerging Non-volatile Memory Based Physical Unclonable Functions," in *2014 IEEE 6th International Memory Workshop (IMW)*. IEEE, May 2014, pp. 1–4.

[15] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "TEAM: ThrEshold Adaptive Memristor Model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 1, pp. 211–221, Jan 2013.

[16] R. Maes, "Physically Unclonable Functions Constructions, Properties and Applications," Ph.D. dissertation, Katholieke Universiteit Leuven, Berlin, Heidelberg, 2013.