Microcontroller Fingerprinting Using Partially Erased NOR Flash Memory Cells

PRAWAR POUDEL, BISWAJIT RAY, and ALEKSANDAR MILENKOVIC,
The University of Alabama in Huntsville

Electronic device fingerprints, unique bit vectors extracted from device's physical properties, are used to differentiate between instances of functionally identical devices. This article introduces a new technique that extracts fingerprints from unique properties of partially erased NOR flash memory cells in modern microcontrollers. NOR flash memories integrated in modern systems-on-a-chip typically hold firmware and read-only data, but they are increasingly in-system-programmable, allowing designers to erase and program them during normal operation. The proposed technique leverages partial erase operations of flash memory segments that bring them into the state that exposes physical properties of the flash memory cells through a digital interface. These properties reflect semiconductor process variations and defects that are unique to each microcontroller or a flash memory segment within a microcontroller. The article explores threshold voltage variation in NOR flash memory cells for generating fingerprints and describes an algorithm for extracting fingerprints. The experimental evaluation utilizing a family of commercial microcontrollers demonstrates that the proposed technique is cost-effective, robust, and resilient to changes in voltage and temperature as well as to aging effects.

CCS Concepts: • **Security and privacy** → *Security in hardware*; **Embedded systems security**; • **Computer systems organization** → *Embedded and cyber-physical systems*; *Embedded systems*; **Embedded software**;

Additional Key Words and Phrases: NOR flash memory, microcontrollers, fingerprinting

ACM Reference format:
Prawar Poudel, Biswajit Ray, and Aleksandar Milenkovic. 2021. Microcontroller Fingerprinting Using Partially Erased NOR Flash Memory Cells. *ACM Trans. Embed. Comput. Syst.* 20, 3, Article 26 (March 2021), 23 pages.
https://doi.org/10.1145/3448271

1 INTRODUCTION

Device identification is one of the key security primitives in integrated circuits, including microcontrollers. A device identifier, also known as a fingerprint, is a bit vector that can be used to differentiate between instances of functionally identical devices. A straightforward approach to device identification is to store a unique device identifier in a dedicated non-volatile memory implemented in either EPROM, EEPROM, flash memory, or programmable fuse. While such identi-

This research was supported in part by the National Science Foundation under Grant No 2007403.
Authors' address: P. Poudel, B. Ray, and A. Milenkovic, Department of Electrical and Computer Engineering, The University of Alabama in Huntsville, 301 Sparkman Drive, Huntsville, AL 35899; emails: {pp0030, biswajit.ray, milenka}@uah.edu.

1539-9087/2021/03-ART26 $15.00
https://doi.org/10.1145/3448271

fiers are reliable and remain stable over time, they can easily be cloned by adversaries. In addition, their implementation increases the device cost due to additional on-chip area required for identifier's storage and interface circuitry. The costs of additional process steps during manufacturing should also be factored in for certain types of non-volatile memories.

A number of alternatives to stored identifiers have been introduced recently. Broadly, they can be classified as either physical fingerprints or semiconductor physical unclonable functions (PUFs) [8]. Whereas physical fingerprints use inherent process variations of integrated circuits to distinguish instances of logically identical chips, PUFs use inherent process variations as a key in mapping input challenges to responses (CR pair) [7]. Gassend et al. introduced the notion of physical unclonable functions in integrated circuits (ICs) and proposed several implementations based on self-oscillating loop with a non-monotonic delay circuit that are tested on FPGAs [4]. Lee et al. fabricated an arbiter-based PUF that exploits inherent delay characteristics of wires and transistors in ICs [14]. Suh and Devadas demonstrated device authentication and secret key generation using ring-oscillator-based PUFs [24]. Whereas these pioneering efforts relied on custom circuits to extract PUFs, a number of alternative approaches emerged that use components that are already in ICs. Thus, Holcomb et al. utilize the states of static random access memory (SRAM) cells after power-up to extract identifying fingerprints of discrete SRAM chips and embedded SRAM modules [8]. Similarly, Liu et al. utilize SRAM cells for creating a PUF using data remanence effect [15]. Bacha and Teodorescu leverage on-chip error correction logic already built into higher-end processor caches [1]. A number of PUF designs leverage unique properties of dynamic RAM memories (DRAMs), including power-up states [27], latency variations [12], write failures [6], refresh pausing [25], or decay [23]. Wang et al. demonstrate that partially programmed commercial NAND flash memory chips can reveal differences in threshold voltages of individual memory cells that are caused by process variations and are unique for each chip [28]. Jia et al. extract keys from NAND flash memory chips using partial erase, partial program, and program disturb techniques [11]. Sakib et al. [22] propose an aging resistant PUF for NAND flash memory utilizing a program disturb method to extract inherent process variations unique to each chip. As for NOR flash memories, Clark et al. leverage erase speed variability in 1.5-transistor flash memory cells for circuit identification [2]. Nguyen et al. [17] propose the use of repeated erase suspend operation for true random number generation and fingerprint extraction using a NOR memory similar to Clark et al. Mandadi [16] relies on repeated partial programming to identify flash memory cells with the minimum threshold voltage in a memory location. The index of the first cell to switch in the memory location is used as a response; responses from multiple consecutive memory locations are concatenated to create a PUF.

Whereas these proposals represent a great advancement in the field of device identification and/or authentication, several important technical problems remain open. First, some of the proposed solutions require dedicated on-chip resources, thus increasing the system cost due to increased chip area and new masks. In addition, they are not applicable to the existing devices. Another class of techniques that utilizes the existing resources often require privileged operations not typically exposed to regular users (e.g., powering cycles, resets, lowering voltage, changing controller parameters) or they require cost-prohibitive computational or storage resources not readily available in low-end computing platforms. Emerging Internet-of-Things applications often rely on resource- and cost-constrained microcontrollers that do not include built-in support for chip identification. However, they often provide high value information and ensuring their authenticity at minimal cost is critical. Unfortunately, the existing proposals for extracting fingerprints from flash memories [2, 16, 17, 22, 28] require computational and RAM memory resources that exceed the capacity of typical low-end microcontrollers.

This article introduces a new method for generating fingerprints in lower-end microcontrollers using partially erased cells of embedded NOR flash memories. Embedded NOR flash memories in microcontrollers are typically used for storing code and data and they are treated as read-only memories. However, modern microcontrollers support in-system programmable flash memories that can be erased and programed internally by running programs. In the proposed method, a flash memory segment erase operation is aborted prematurely after a predefined time, so that its state reflects physical cell-to-cell variations, which are unique for each device or a segment within a flash memory. The proposed method for extracting a flash memory fingerprint that can serve as a device fingerprint is highly reliable, requires lightweight compute and modest storage resources, and is thus highly suitable for low-end microcontrollers. It does not require error detection and correction algorithms, thus minimizing compute requirements, and maps one bit of flash memory to one bit of fingerprint.

The key contributions of this article are as follows: (a) it characterizes behavior of embedded NOR flash memories when subjected to partial erase operations; (b) it introduces a technique for extracting device fingerprints during enrollment and authentication phases; (c) it explores the robustness of the proposed technique as a function of the operating conditions and state of the chip.

The rest of the article is organized as follows. Section 2 gives background on device fingerprints and discusses NOR flash memory preliminaries. Section 3 describes NOR flash memory characterization and how to extract physical properties of flash memory cells through a digital interface. Section 4 describes a method for creating NOR flash fingerprints using a partial erase operation. Section 5 describes the proposed algorithms for device enrollment and authentication. Section 6 presents the setup used in the experimental evaluation and the results of the experimental evaluation. Section 7 gives a brief overview of the related work and Section 8 concludes the article.

2 BACKGROUND

This subsection first gives a brief overview of fingerprint-based authentication. Then, it discusses organization and operation of embedded NOR flash memories necessary for understanding of the proposed mechanism.

2.1 Device Fingerprints

PUFs are one-way functions that respond to an input challenge by providing an output response. CR pairs, used to uniquely identify and authenticate devices, are derived from different physical properties, including timing delays in oscillating loops [4], delays in transistors and wires [14], delays in oscillators [24], power-up state of SRAMs [8], remanence effect of SRAM [15], different properties of DRAMs [6, 12, 20, 22, 24–26], and threshold voltage variation of flash memories [2, 11, 16, 19, 21, 27]. All of these reflect unintended manufacturing variations in functionally identical devices. Since these variations among the devices are inherent, they cannot be cloned or induced intentionally into the devices in question.

Device fingerprints are also derived from the physical properties of the electronic devices. However, unlike PUFs, extraction of device fingerprints requires fewer resources; but at the same time, they are affected by environmental factors like temperature and noise. Thus, fingerprint-based authentication relies on similarity within a range of an established metric, such as Hamming Distance [6, 8, 16, 21, 22, 24, 26], Jaccard Index [12], and Correlation Value [19, 27]. The term PUFs and device fingerprints are sometimes used interchangeably in the open literature.

Figure 1 illustrates a system view of the fingerprint-based device authentication. It includes two phases, *enrollment* and *authentication*. The enrollment is typically conducted at the manufacturer's site before devices are shipped. The manufacturer characterizes devices and extracts

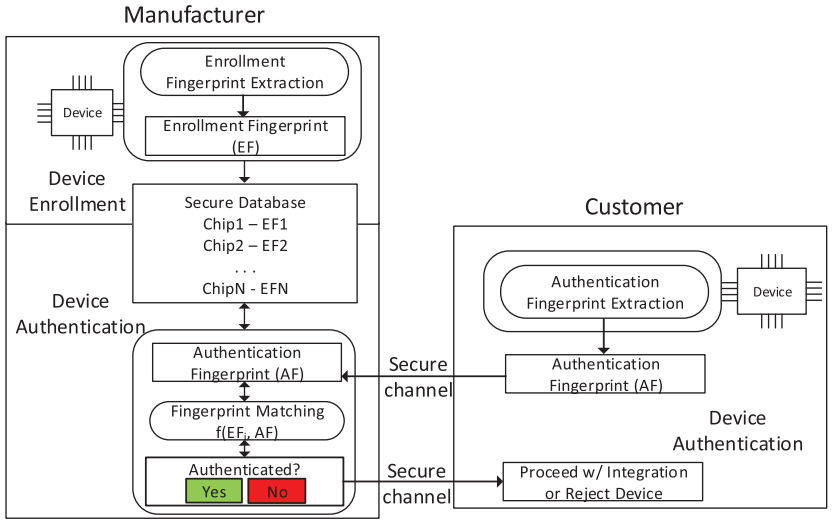

Fig. 1. System view of a fingerprint-based authentication system.

fingerprints—bit vectors unique to each device extracted from a chosen physical property. The fingerprints generated in the enrollment phase are called *Enrollment Fingerprint (EF)*. These fingerprints are stored in a secure database in a manufacturer's server as shown in Figure 1. Secure authentication services are built on top of the secure database and are used by customers during authentication phase.

The authentication involves steps performed at the customer's site and the manufacturer's site. Before integrating devices into their products, the customer extracts a device fingerprint using a known algorithm publicized by the manufacturer. This fingerprint is known as Authentication Fingerprint (AF). A secure communication channel is established between the customer and the manufacturer's device authentication service. The customer sends the authentication fingerprint that is verified at the manufacturer's site by comparing the authentication fingerprint to the enrollment fingerprints in the database. The fingerprint matching relies on one of the metrics, such as Hamming Distance, Jaccard Index, or Correlation Value. Based on the matching criteria, the manufacturer's authentication service generates a response to the customer to either confirm or deny authenticity of the device.

Whereas the scenario described above discusses the use of fingerprints to ensure authenticity of semiconductor devices and thus prevent counterfeits, it can also be used in other scenarios. For example, an IoT platform vendor can build an authentication service using fingerprints of individual physical devices of each platform to control access to its service. This way only genuine platforms can access a service such as firmware update.

2.2 Flash Memory Cell: A Physical View

Flash memories are non-volatile memories that store information in an array of memory cells made from floating gate transistors. A floating-gate transistor is an NMOS transistor that consists of two stacked gates – the regular control gate (CG) that sits on the top and the floating gate (FG) that is sandwiched between the control gate and the transistor's conductive channel formed between source (S) and drain (D) terminals in substrate (Figure 2(a)). Figure 2(b) shows a symbol for a floating-gate transistor. The floating gate, surrounded by oxide and electrically insulated from the transistor terminals, can trap negative charge that stays on it even when power supply is turned

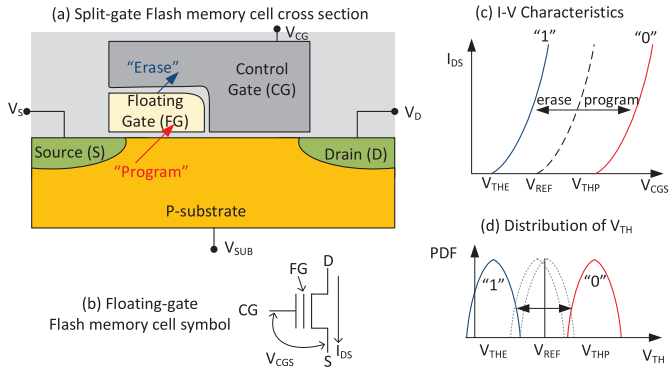

Fig. 2. (a) Cross-section of split-gate flash memory cell [4]. (b) Floating-gate transistor symbol. (c) Current-voltage characteristics of a split-gate/flash memory cell (V_{THE}–threshold voltage for the erased state, V_{THP}–threshold voltage for the programmed state, V_{REF}–reference threshold voltage); (d) Probability density function of the threshold voltage for erased, programmed, and unstable cells.

off. Trapped negative charge on the floating gate effectively increases the transistor's threshold voltage. A flash memory cell typically keeps one bit of information (single-level cells or SLCs), though multi-level (MLCs) and triple-level cells (TLCs) are used in high-density flash memories for storing two and three bits in a single cell, respectively. An SLC flash memory cell is in a so-called erased state (reads as a logic "1") when there is no trapped negative charge on the floating gate, whereas it is in a programmed state (reads as a logic "0") when there is negative charge. Figure 2(a) shows a cross-section of a split-gate flash memory cell that keeps one bit of information [4]. This arrangement slightly differs from a traditional stacked flash memory cell. Here, the floating gate occupies a portion of the area between the drain and source. The control gate has a unique shape that covers a portion of the area between the drain and source as well as a portion of space above the floating gate [3, 9].

To change the state of a flash memory cell, two operations are performed: program and erase. The program operation charges the floating gate with electrons, whereas the erase operation removes the charge from the floating gate. These operations require high voltages and are carried out through the oxide as shown in Figure 2(a). To program a split-gate flash memory cell, a large voltage is applied to the source terminal (V_S~10V, V_{CG}~2V, V_D~0.5V), inducing source-side hot carrier injection (SSI) that traps electrons on the floating gate ("Program" arrow). The negative charge on the floating gate reduces the voltage between the control gate and the source, thus increasing the threshold voltage (V_{TH}=V_{THP}) as shown in Figure 2(c). To erase flash memory cells, a large positive voltage is applied on the control gate (V_{CG}~12V, V_S=V_D=0V) to remove electrons from the floating gate via Fowler-Nordheim tunneling ("Erase" arrow). The removal of electrons decreases the threshold voltage (V_{TH}=V_{THE}) as shown in Figure 2(c). A read operation from flash memory involves applying a read voltage on the control gate, V_{CG}=V_{READ}~3V, and a sense voltage on the drain, V_D=V_{SENSE}~2V, and sensing the threshold voltage. An erased cell conducts the current and that is sensed as a logic "1." A programmed cell does not conduct the current and that is sensed as a logic "0."

2.3 Flash Memory Organization

Depending on the way memory cells are organized in arrays of cells, we distinguish between NOR and NAND flash memories. NOR flash memories are designed to allow random access through full address and data buses. Fast reads, low standby power, and robust operation make them an

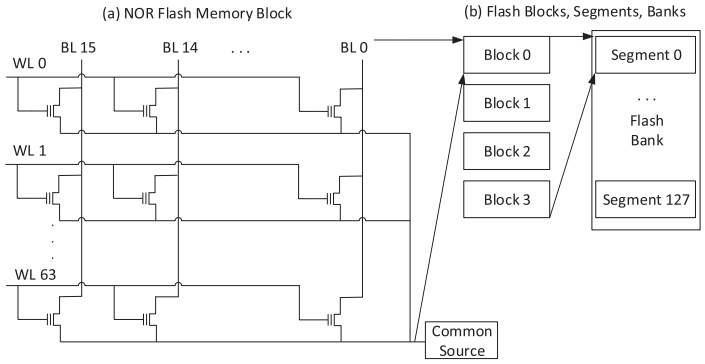

Fig. 3. (a) NOR flash memory block architecture. (b) Flash memory organization: blocks; segments, and banks.

ideal choice for storing code and data in embedded systems and systems-on-a-chip. However, cells in NOR flash memories are larger than those in NAND flash memories, resulting in lower density and higher cost per bit and longer erase and program times. In contrast, NAND flash memories are designed for high-capacity and low-cost storage solutions (e.g., solid-state disk drives), but they do not provide a random-access address bus. In systems that use NAND flash for storing program code, programs are typically executed by shadowing the contents to a RAM memory. In this article, we focus on NOR flash memories, specifically NOR flash memories embedded in modern SoCs.

NOR flash memory cells are organized in two-dimensional arrays known as flash memory blocks as shown in Figure 3(a) [9]. The control gates of cells in each row are electrically connected through a line called Word Line (WL). The drains of all cells in a single column are electrically connected through a single line called Bit Line (BL). The source terminals of all cells are also electrically connected to a common source terminal. The number of cells in each row defines a word size; here, we assume 16-bit words. Thus, a flash memory block illustrated in Figure 3(a) has capacity of 64 16-bit words or 128 bytes. Blocks are further organized into segments. In our example shown in Figure 3(b), a segment includes four blocks for a total of 256 words (i.e., $N_{words} = 256$). Segments are further organized into flash memory banks, e.g., a 64 KB flash memory bank includes 128 segments.

Flash memory reads are performed at a byte or a word granularity as all the bits in a word share the same WL. The sensing process involves application of a sense voltage V_{SENSE}~2V on the bit lines and a read voltage V_{CG}~3V on the selected word line (the common source is grounded, V_S=0V). All other unselected WLs are biased at low voltage so that unselected memory cells are turned off. With this biasing arrangement, current will flow from the bit line to the ground through the bits whose threshold voltage is less than the read voltage. The read reference voltage (V_{REF}) is set between the erased state and programmed state threshold voltage distributions, so that there is enough of noise margin to correctly identify the bit states as shown in Figure 2(d).

2.4 Flash Memory in Microcontrollers

Modern microcontrollers targeting low-cost and low-power applications are typically designed as a system-on-a-chip that integrates a processor core, flash memory, RAM memory, clock subsystems, and a number of input/output peripherals, such as parallel ports, timers, serial communication interfaces, ADCs, DACs, and others. As non-volatile microcontroller memories, flash memories are used to store code and data constants. In the default mode of operation, the microcontroller flash memory behaves as a ROM, allowing instruction fetches and data reads at a byte or word level. However, flash memories are often in-system programmable through a flash memory

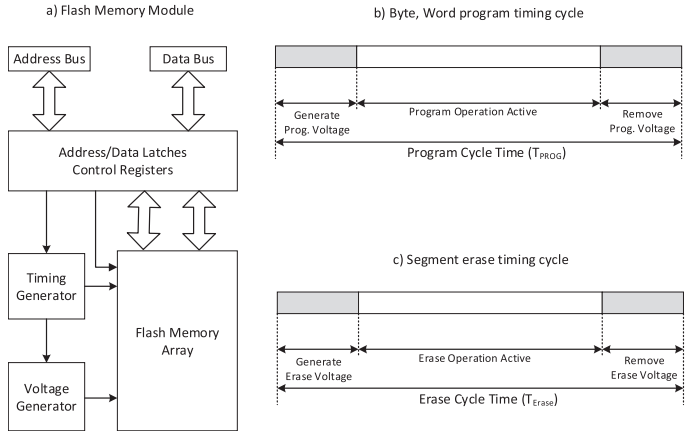

Fig. 4. (a) Block diagram of a flash memory module. (b) Program Cycle Time. (c) Erase Cycle Time.

controller that supports programming individual bits, bytes, or words. Each flash memory bit can be programmed from logic "1" to logic "0" individually, but an erase operation is required to reprogram a bit from logic "0" to logic "1." In other words, overwriting an existing flash content does not work without an intermediate erase operation. Flash memory controllers typically support an erase operation that will erase an entire flash segment and often a mass erase operation that will erase an entire flash memory bank.

Figure 4(a) shows a block diagram of a flash memory module in a microcontroller that includes a flash memory with one or more banks and a flash memory controller. The controller includes voltage generators that supply voltage levels needed for program and erase operations, as well as timing generators that control duration of operations that span multiple clock cycles. A flash memory program (write) operation can be initiated by a microcontroller code that is running from within the flash memory itself or from RAM. When initiating a write operation from within the flash memory, the processor is typically halted until the program operation completes. When a flash operation is initiated from RAM memory, the processor can continue executing the code from the RAM, provided it does not interact with the flash memory bank that contains a location currently being programmed.

Figure 4(b) shows a typical program cycle that encompasses times to bring up voltage generators, perform the program operation, and remove programming voltages to allow flash memory to resume operation in its default mode. The total programming time in a microcontroller used in our study is $T_{PROG} \approx 64–85\ \mu s$ [26]. Similar to program operations, erase operations can be initiated from within the flash memory halting the processor, or from within the RAM allowing the processor to continue execution. Figure 4(c) shows a typical erase cycle that encompasses times to bring up voltage generators, perform segment erase operation, and remove programming voltages to allow flash memory to resume operations in its default mode. The total segment erase time in a microcontroller used in this study is $T_{ERASE} \approx 23–35$ ms [10].

2.5 Partial Flash Erase

As in-system flash program and erase operations take multiple processor clock cycles, flash memory controllers often allow for issuing an emergency exit that cancels ongoing flash operations, leaving flash cells in an unknown state. The emergency exit during flash program/erase operations, i.e., partial program/erase operations, may result in unstable states of flash memory cells. Poudel et al. used partial program operations to induce perturbed states of memory cells [19], that are in

```
1.  uint_16 *pFlashAdr = SEGMENT_ADR;
2.  void PartialEraseSegment(uint16 TPERASE) {
3.    while (FCTL3&BUSY); // Wait if BUSY
4.    FCTL3 = FWPW;       // Clear LOCK bit
5.    FCTL1 = FWPW+ERASE; // Enable erase operation
6.    *pFlashAdr = 0;     // Dummy write (0) to initiate erase
7.    Delay(TPERASE);     // Software delay for TPERASE
8.    FCTL3 = FWPW+EMEX;  // Emergency Exit
9.    FCTL1 = FWPW;       // Clear ERASE bit
10.   FCTL3 = FWPW+LOCK;  // Set LOCK
11.   while(FCTL3&BUSY);  // Wait if BUSY
12. }
```

Fig. 5. C subroutine for partially erasing a flash memory segment.

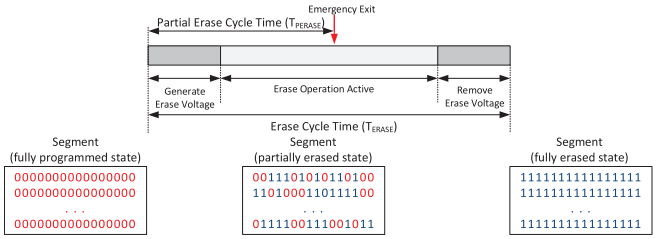

Fig. 6. Partially erased flash memory segment.

turn used as a source of randomness in generating true random numbers. However, in this article, we utilize partial erase operations of flash memory segments to create unique device fingerprints by distinguishing flash memory cells that quickly change their state from programmed to erased from those that require more time. Although partial program operations can also be used for this purpose, this method is not optimal as it requires more time and a greater precision in timing partial flash operations. A NOR flash program operation is carried out on bytes or 16-bit words. To generate long bit vectors, a partial program operation needs to be repeated on multiple words to achieve desired fingerprint length. On the other side, a NOR flash erase operation is carried out over an entire segment (4 Kbits in our case), thus providing a sufficient number of bits for extracting a fingerprint. In addition, flash erase operations take more time than program operations, making it easier to find an optimal duration of the partial operation.

Let us first describe how a partial erase operation is carried out. A chosen flash segment is first erased and then fully programmed, so all memory bits within the segment will read as "0." Next, a subroutine for partial erase of the segment is invoked (Figure 5). It initializes the flash controller for the erase operation (lines 4 and 5 in Figure 5). Any memory writes to any of the addresses that belong to the segment will initiate an erase operation (line 6). Typically, during the erase operation the flash memory is not responsive, stalling program execution. However, if the code is executed from the RAM memory or a different flash bank, then the processor can continue execution. Line 7 invokes parametrized software delay before aborting the ongoing erase operation. In the microcontroller used in this study, the erase operation is aborted by simply setting the emergency exit bit (EMEX) in the control register of the flash controller (line 8). By tuning the parameter T_{PERASE}, we can control the duration of the partial erase down to the granularity of a single processor clock cycle.

To create a unique fingerprint from a memory segment, we need to abort the erase operation at the point where the number of bits in the erased state is roughly equal to the number of bits in the programmed state (Figure 6). We argue that a fingerprint created this way captures the physical properties of the flash memory segment due to manufacturing variations that are unique

```
1.  T_PERASE = 0;
2.  do {
3.    FullEraseSegment;              // Write 1s
4.    FullProgramSegment;            // Write 0s
5.    PartialEraseSegment(T_PERASE); // Partial erase
6.    FS_AND=11..1b; FS_OR=00..0b;
7.    for(i=0; i<N_R; i++){          // N_R segment reads
8.      F_R = ReadEntireSegment;
9.      FS_AND &= F_R;
10.     FS_OR  |= F_R;
11.     SWDelay();
12.   }
13.   Stable1s = FS_AND & FS_OR;
14.   Stable0s = FS_AND | FS_OR;
15.   UnstableBits = FS_AND ^ FS_OR;
16.   Ratio = Count(Stable1s)/Count(Stable0s);
17.   T_PERASE += Δt;                // Increase T_PERASE
18. while (T_PERASE <= T_ERASE);
```

Fig. 7. Characterization of partial erase operations of a segment.

for each segment, i.e., there are no two segments on the same chip or different chips that will have identical or similar fingerprints. The next section describes characterization of partially erased flash memory segments that supports the above argument.

3 FLASH MEMORY PARTIAL ERASE CHARACTERIZATION

To characterize the process of partially erasing a flash memory segment, we perform the following steps, illustrated in Figure 7. First, the segment is fully erased (line 3) and fully programmed (line 4). Next, the partial erase operation is initiated. To observe its effects, the flash memory is repeatedly read a certain number of times (N_R). The state of individual bits is tracked; the bits are classified to those that are stable programmed ("0" bits), stable erased ("1" bits), and those that are unstable, i.e., their value sometimes reads as a logic "0" and sometimes as a logic "1." The state of the partially erased segment can be described by a vector of bits with values "0" for stable programmed bits, "1" for stable erased bits, and "x" for unstable bits. Stable "0" and "1" bits as well as unstable bits are counted, and these numbers are reported. Finally, the partial erase time is increased for a certain period (Δt) and the process is repeated until the partial erase time matches the nominal erase time specified for the given flash memory. The initial duration of the partial erase operation is set to T_{PERASE}=0 s.

Figure 8 shows the results of the characterization for 12 flash memory segments that are collected from 3 sample microcontrollers. The red lines show the percentage of the stable programmed bits, the blue lines show the percentage of stable erased bits, and the green lines show the percentage of unstable bits as a function of the partial erase time (T_{PERASE}). We can identify three distinct regions. In the first region where T_{PERASE} is very small, all bits remain in the programmed state (red lines are at 100% and blue lines are at 0%). The second region is characterized by a steep slope where a small change in the partial erase time dramatically impacts the state of the flash cells. We can identify a partial erase time window where the number of stable "0"s and the number of stable "1"s intersects. Please note that the number of unstable bits peaks in this region too, but their share never exceeds 2% of all bits. Finally, in the third region, the majority of bits are in the erased state (>85%), and the number of erased bits is slowly approaching 100% as we increase T_{PERASE}. Please note that Figure 8 shows the state of the flash memory cells for the first 0.2 ms of the partial erase time, whereas the total measured segment erase time with no emergency exit is ~25 ms.

From this analysis, we can conclude that by carefully tuning the partial erase time, we can bring the flash memory cells into the state where approximately half of the bits are stable "0"s and other

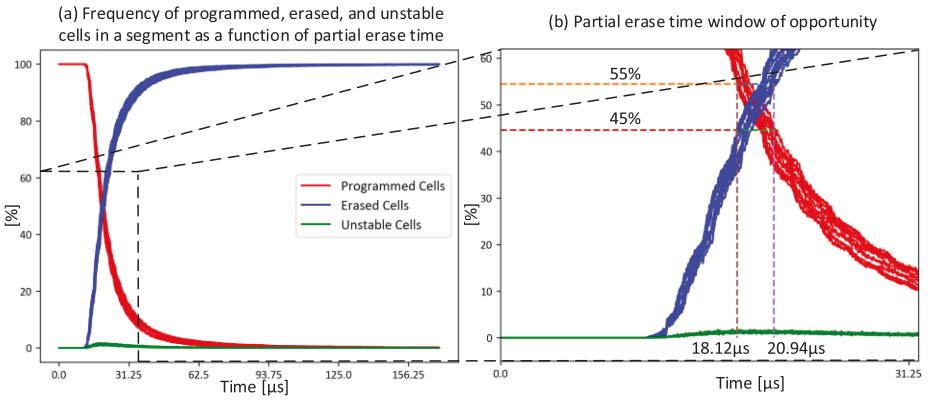

Fig. 8. The frequency of programmed, erased and unstable bits as a function of the partial erase time. (T_{PERASE}).

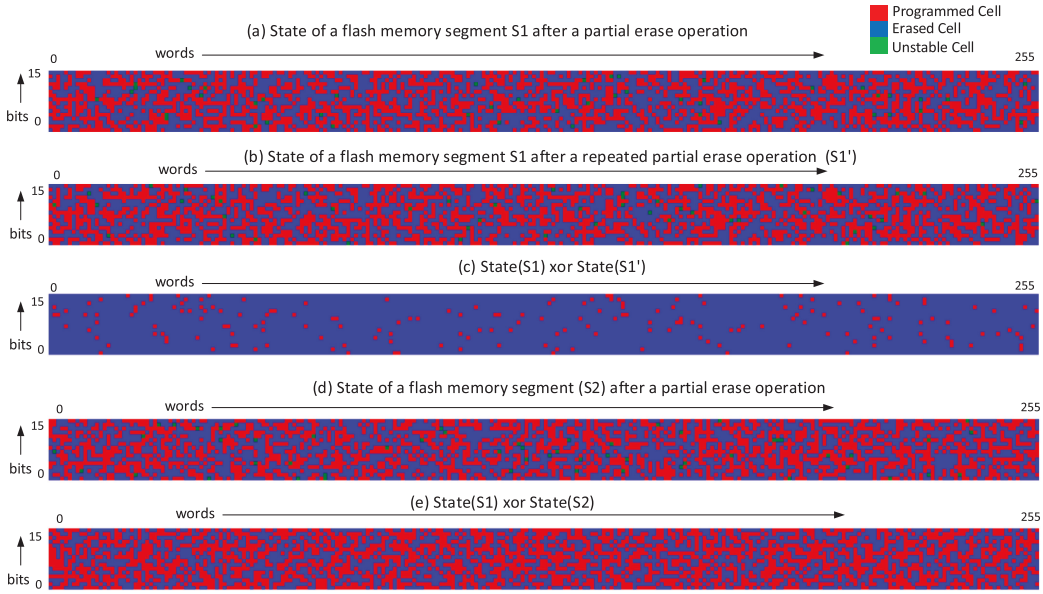

Fig. 9. State of partially erased flash segments.

half are stable "1"s, with a tiny percentage of unstable cells. In the next section, we will describe why these properties are suitable for generating fingerprints.

4 FLASH MEMORY FINGERPRINTS

Let us consider a flash segment that is partially erased using steps described in Figure 5 so that the number of stable "0"s is approximately equal to the number of stable "1"s. Figure 9(a) illustrates the state of such a flash memory segment (S1) with 256 16-bit words. The distribution of the stable programmed and erased cells appears to be random and the number of unstable cells is relatively small. If we repeat the characterization procedure of the segment S1 (erase the segment, fully program it, partially erase it using the same or similar T_{PERASE}, and then characterize the cells by repeated reading), then we get a new state of the flash segment shown in Figure 9(b) (let us call this

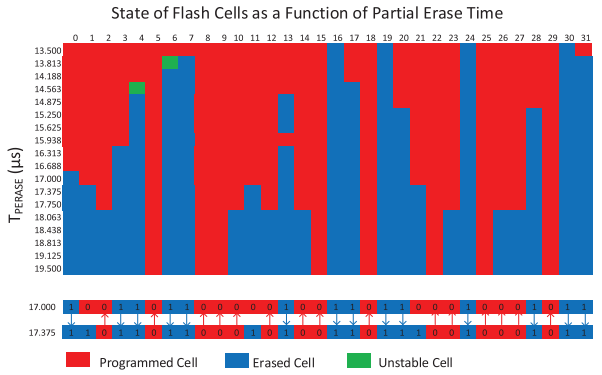

Fig. 10. States of flash memory cells as a function of partial erase time.

state S1'). Figure 9(c) shows the result of xoring two states of the same segment. A vast majority of the bits has the same state after a repeated partial erase (the area is predominantly blue).

On the other side, by repeating these steps on a different flash segment in the same chip or in a different chip, we will get the flash state that is remarkably different from the ones observed in Figures 9(a) and 9(b). For example, Figure 9(d) shows the state of a partially erased flash segment S2. To underscore its difference relative to the segment S1, Figure 9(e) shows the result of xoring the states of segments S1 and S2.

Whereas Figure 9 shows that the state of flash memory segments for a given T_{PERASE} is repeatable and unique, we would like to see how the state of the flash cells within a segment changes as a function of T_{PERASE}. To examine this property, multiple experiments are conducted by repeating the characterization procedure while the partial erase time is increased in each round of the experiment. Figure 10 illustrates the state of a 32-bit double word in a flash segment as we increase T_{PERASE} step-by-step from 13.5 to 19.5 μs. We can observe that each flash cell changes its state from programmed to erased at a particular erase time. The transitions exhibit monotonicity – the memory cells typically remain in the erased state once they change the state. Exceptions are possible, but they are rare, e.g., bit 13 in Figure 10. Let us consider the state of this 32-bit double word for T_{PERASE}=17.375 μs. The number of erased cells is 17 out of 32 (53.125%), the number of programmed cells is 15, and there are no unstable cells. Let us consider this bit vector as a portion of the established device fingerprint or enrollment fingerprint. Any future identification process that repeats the steps to bring the flash segment into a partially erased state with $T_{PEARSE} \approx 17.375$ μs should yield a similar fingerprint. Likewise, a fingerprint captured on a different logical device should be quite different from this fingerprint.

Let us develop a metric for comparing the enrollment fingerprint (EF) and authentication fingerprints (AF). Let us further assume that during authentication slightly different T_{PEARSE}= 17.0 μs is used ($T_{PEARSE.AUTH} < T_{PERASE.ENROL}$). In this case, we expect that any programmed bit that was observed in the enrollment fingerprint should remain programmed in the authentication fingerprint, because $T_{PERASE.AUTH} < T_{PERASE.ENROL}$. Similarly, any erased bit in the authentication fingerprint should be found erased in the enrollment fingerprint. To capture these properties, we can define the following parameters:

(a) **Matching "0"s**: Matching "0"s (or programmed bits) in EF and AF should be a subset of "0"s in EF, i.e., the ratio of the number of matching 0s in EF and AF to the number of 0s in EF should be close to 1 in an ideal case.

```
1.  done = false;
2.  T_PERASE = (T_PERASE.min + T_PERASE.max)/2;
3.  do {
4.    fingerprint = 0000..00b;   // Initialize vector fingerprint
5.    count_erased = 0;          // Clear the number of erased bits
6.    FullEraseSegment;
7.    FullProgramSegment;
8.    PartialEraseSegment(T_PERASE);
9.    for(i=0; i<N_Words; i++){     // For each word in a segment
10.     FS_SUM[16]= {};             // Initialize bit sum vector
11.     for(j=0; j<N_R; j++){       // For N_R times
12.       F_R = ReadWord(i);        // Read word i
13.       FS_SUM += F_R;            // Sum individual bit values
14.     }
15.     for(b_index=0; b_index<16; b_index++){ // For each bit
16.       if(FS_SUM>N_R/2) {                   // Use majority voting
17.         fingerprint |= (1<<(b_index*i));   // to determine its value
18.         count_erased++;
19.       }
20.     }
21.   }
22.   Ratio = (count_erased)/(4096);
23.   if (Ratio > 0.50 && Ratio <= 0.55) done = true;
24.   else if (Ratio < 0.50) T_PERASE = T_PERASE + Δt;
25.   else T_PERASE = T_PERASE - Δt;
26. } while (!done);
27. EnrollmentFingerprint = {fingerprint, SegmentAddress};
```

Fig. 11. Algorithm for generating enrollment fingerprint.

(b) **Matching "1"s**: Matching "1"s (or erased bits) in EF and AF should be a subset of "1"s in AF, i.e., the ratio of the number of matching "1"s in EF and AF to the number of "1"s in AF should be close to 1 in an ideal case.

Thus, by combining these two parameters, we derive the Similarity Index, SI, as follows:

$$SI = \frac{\left(\frac{\#Matching\ 0s\ in\ EF\ and\ AF}{\#0s\ in\ EF} + \frac{\#Matching\ 1s\ in\ EF\ and\ AF}{\#1s\ in\ AF}\right)}{2} \tag{1}$$

In the example from Figure 10, the enrollment fingerprint has 17 "1"s and 15 "0"s, whereas the authentication fingerprint has 14 "1"s and 18 "0"s. Between the fingerprints, the number of matching "1"s is 14, and the number of matching "0"s is 15. Thus, the similarity metric is (14/14 + 15/15)/2 = 1. Thus, these two fingerprints are considered fully matching.

5 DEVICE IDENTIFICATION USING FLASH MEMORY FINGERPRINTS

In this section, we describe how flash-memory fingerprints derived using the method described above can be used to verify authenticity of devices. The protocol involves two steps: (a) device enrollment and (b) device authentication similar to the system view presented in Section 2.1. Here, we elaborate the protocol specific to the proposal presented in this article.

Device enrollment phase that is carried out by either the device manufacturer (chip maker) or the manufacturer of the product (e.g., IoT platform vendor) results in the enrollment device fingerprint that is stored in a database. The algorithm for fingerprint enrollment is shown in Figure 11. It is remarkably similar to the algorithm used to characterize the flash segment shown in Figure 7. Instead of walking through all partial erase times, the algorithm searches for a suitable T_{PERASE} within a time window [$T_{PERASE.MIN}$, $T_{PERASE.MAX}$], most likely to result in evenly split number of programmed and erased bits. The selected flash segment is first fully erased and fully programmed (lines 6 and 7), partially erased (line 8), and then characterized by repeated reading (lines 11–14). The parameter N_R represents the number of consecutive reads. The flash segment bits are

classified as either programmed or erased using majority voting (lines 15–19). For the bit position value read as a logic 1 for more than $N_R/2$ times, the corresponding fingerprint bit is set to 1 (line 17). Since the vector *fingerprint* is initialized to all "0"s, no action is need for bit position that is classified as programmed. The ratio of the number of erased bits to the total number of bits in the segment is also determined (line 22), and if it is in the range [50%, 55%], the current value of bit vector *fingerprint* is recorded as enrollment fingerprint. Optionally, the enrollment fingerprint descriptor could include the address of the flash segment, if multiple segments are used. If the number of erased bits is outside the desired range, then the partial erase time is increased or decreased, depending on its observed value (lines 24 and 25), and the steps are repeated with the new partial erase time. In case of a microcontroller used in this research, the partial erase time can be tuned with the resolution of a single processor clock cycle, though course-grained time steps would suffice. The algorithm efficiency improves by minimizing the number of tries to get a valid fingerprint. Please note that in the interest of simplicity, the proposed algorithm does not consider unstable bits, because they are remarkably rare, as discussed in Section 3.

It should be noted that multiple fingerprints can be created for a single microcontroller, one per each flash memory segment. Alternatively, only a portion of a flash segment can be deemed sufficient for creating fingerprints. At the moment, we assume that all 4,096 bits in a segment are used as a fingerprint. However, later we will analyze shorter fingerprints with 2,048, 1,024, 512, and 256 bits.

The device authentication is initiated by the customer side. We assume that the lower boundary of the Ratio parameter used in creating enrollment fingerprints (0.5 in Figure 11) as well as typical $T_{PERASE.ENROLL}$ for given family of devices are publicly disclosed and known to the customers. Alternatively, the customer may place an authentication request for the given type of devices and the manufacturer responds by providing these parameters. In case that multiple enrollment fingerprints exist for a single device, the manufacturer may also send the starting address of a flash memory segment to be used for generating the authentication fingerprint. Then the customer extracts the authentication fingerprint from the desired segment by executing the following steps: (a) fully erase and program it; (b) partially erase the segment using $T_{PERASE} = T_{PERASE.ENROLL} - dT$; (c) obtain the authentication fingerprint by using the repeated reading of the flash segment – the same steps used in the enrollment (lines 9–21 in Figure 11); (d) if the $0.45 \leq Ratio_{AUTHENTICATE} \leq 0.5$, then use the fingerprint from (c); otherwise, adjust the partial erase time and repeat the steps until this condition is satisfied. The authentication fingerprint is then sent to the manufacturer that compares the enrollment and authentication fingerprints. If they meet the similarity metric (e.g., $SI > 0.85$), then the device is deemed authentic. Otherwise, the device fails the authentication.

6 RESULTS

6.1 Experimental Setup

Our experimental setup is based on MSP430, a mixed-signal microcontroller family from Texas Instruments. The MSP430 microcontrollers are built as a system-on-a-chip, integrating a 16-bit processor core, flash memory, SRAM memory, clock oscillators, and a wide range of 8-bit and 16-bit input/output peripherals, including parallel ports, timers, comparators, analog-to-digital and digital-to-analog converters, serial communication interfaces, LCD controllers, and DMAs. MSP430 devices differ in processor speed, size of memories, and the number and type of peripherals. The clock subsystem is controllable from software and supports several clock signals that can be changed or selectively turned on and off to allow for a low power operation. The flash memory is in-system programmable and its architecture corresponds to the one described in Sections 2.3 and 2.4.

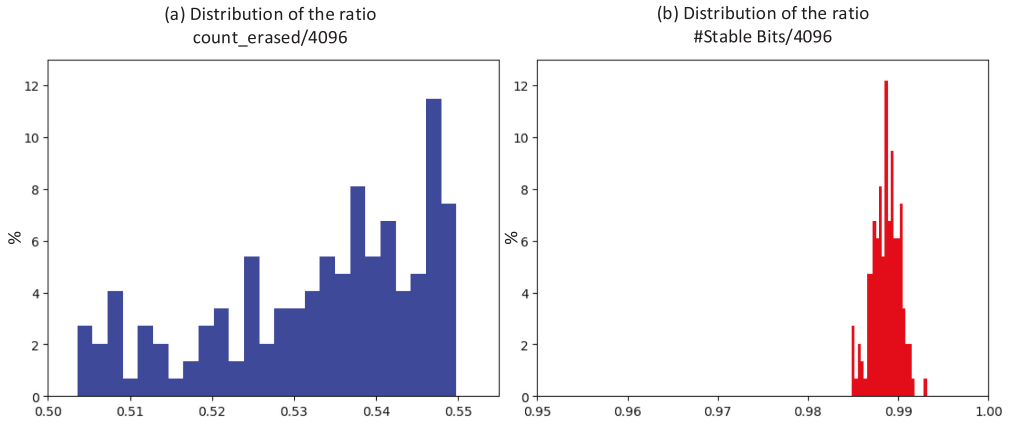

Fig. 12. Fingerprint enrollment characterization.

In this study, we use MSP430F5438 that includes 256 KB of flash memory (4 banks with 64 KB each), 16 KB of SRAM memory, timers, serial communication interfaces, parallel ports, and an ADC controller. The experiments are conducted using the Experimenter Development Board, EXP430F5438. It features a 100-pin drop-in socket for microcontrollers allowing for quick changes of microcontroller chips. Three physical devices are used in the experiments, each with 50 flash memory segments, for a total of 150 logical devices.

6.2 Evaluation

Figure 12 shows the results of enrollment characterization. Figure 12(a) shows the distribution of the ratio of the erased cells ("1") to the total cells in a segment. The target ratio is between 50% and 55% (see Figure 11). As our logical devices showed rather similar characteristics, the successful enrollment can be found with a minimum number of tries. There is a trade-off between the minimum change of the partial erase time (Δt parameter in Figure 11) and the ability to achieve the target ratio. Though our setup allows to control the partial erase time with resolution of a single clock cycle (~16 MHz) and thus achieve the ratio of ~1 (the number of 1s is roughly equal to the number of 0 s), we used a relatively coarse grained Δt to minimize the number of steps during enrollment phase. As our logical devices had remarkably similar characteristics (see Figure 8), our enrollment typically takes place on a first try. For each enrollment fingerprint, we ensure that the number of unstable bits is below a certain threshold (2%). Figure 12(b) shows the distribution of the stable bits in a segment—more than 98% of all bits in fingerprints are characterized to be stable. This experiment confirms that the number of unstable bits during the partial erase process does not exceed the given threshold.

Figure 13 shows the results of the similarity index evaluation. In this experiment, we determine authentication fingerprints for all logical devices. The authentication fingerprints are compared against all other enrollment fingerprints. Figure 13(a) shows the distribution of the similarity index when an authentication fingerprint for the given logical device is compared to enrollment fingerprints of all other logical devices (inter similarity index as a measure for *Uniqueness*). The inter similarity index peaks at 0.5 and ranges between 0.47 and 0.53, which ought to be adequate. Similarly, Figure 13(b) shows the similarity index determined by pairing the corresponding authentication and enrollment fingerprints of all logical devices (self-similarity index as a measure of *Reliability*). The self-similarity index ranges between 0.89 and 0.97. Consequently, we can establish threshold for authentication process—if the similarity index of the given authentication

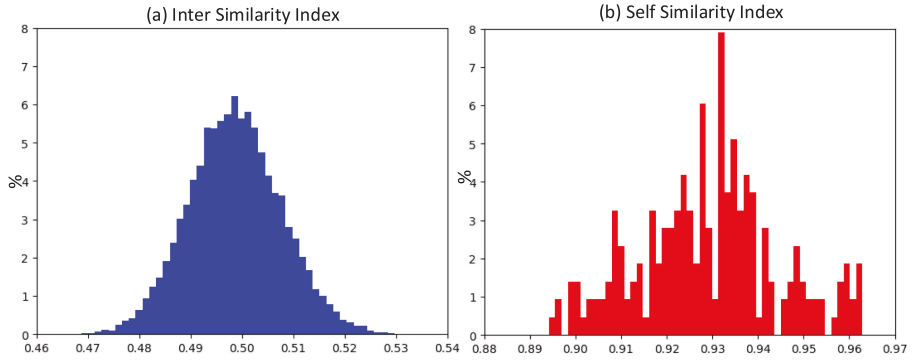

Fig. 13. Evaluation of Similarity Index.

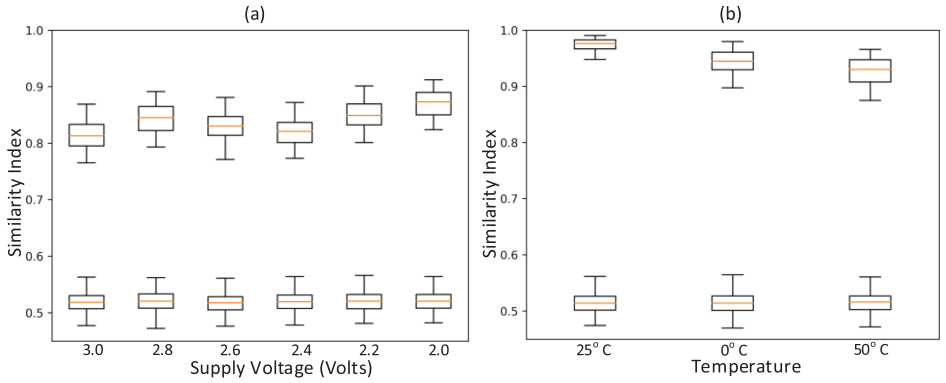

Fig. 14. Impact of power-supply on Similarity Index (a) and impact of ambient temperature on Similarity Index (b).

fingerprints is at least 0.89 to the one of the enrolment fingerprints in the database, then the device passes the authentication. Otherwise, the authentication fails.

6.3 Environmental Factors

For the proposed method to be a viable solution for device authentication, it needs to be resilient to environmental conditions, namely, to the changes in the power supply and ambient temperature. To evaluate impact of these two, we conduct experiments where authentication fingerprints are collected under different environmental factors. We assume that the enrollment fingerprints are created under default power supply at room temperature (V_S=3.3 V, T=25°C).

Figure 14(a) shows the inter- (bottom) and self- (top) similarity index distributions as a function of power supply voltage varied between 3.0 and 2.0 V (lowest recommended voltage for the flash memory operations to work reliably). The plot shows the median (orange horizontal line), Q1–Q3 quantiles, and the minimum and maximum of the similarity indices. The results show that changes in the power supply voltage during authentication have relatively little impact on the inter-similarity index if we are in the recommended operating range. Whereas the changes in the self-similarity index are noticeable, they are still not significant to impact the viability of the proposed mechanism, the distance between two distributions remain high. More importantly, we have not observed any trends in the similarity index as a function of the power supply. The reason for this is that the internal charge pumps are providing relatively stable voltage to the flash memory

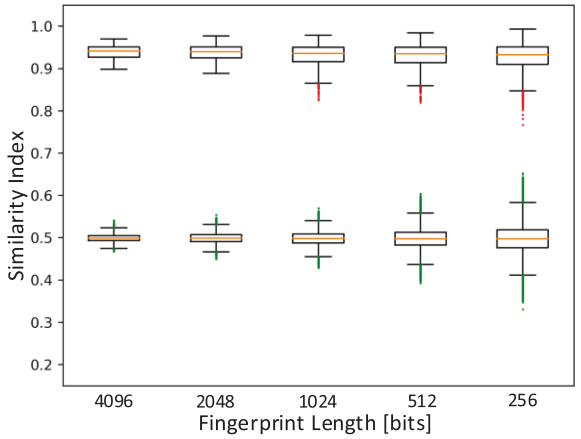

Fig. 15. Similarity Index evaluation for sub-segment fingerprints.

regardless of the power supply, as long as it is in a given range. It should be noted that changes in the power supply may impact the frequency of internally generated processor clocks that in turn impacts the partial erase time. However, the proposed authentication method allows for adjusting the partial erase time, so that the authentication fingerprint meets qualification criteria.

Figure 14(b) shows the inter- (bottom) and self- (top) similarity index distributions as a function of ambient temperature. The experiments are conducted on a subset of logical devices. The results of experiments conducted at the room temperature are in the first column (median, Q1–Q3, range). Again, the authentication fingerprints collected at different ambient temperatures (at 0°C and 50°C) are compared against the corresponding enrollment fingerprints collected at the room temperature. We can see that the impact of temperature on the inter-similarity index is minimal. It does reduce the median of the self-similarity index and its distribution widens, but again the distance between the inter- and self-similarity indices remains healthy. Our setup is not conducive to testing wider temperature ranges, but we do not expect them to impact viability of the proposed mechanism.

Overall, we find these results acceptable as authentication conditions are typically known when going through the authentication step. Specifically, the authentication may be modified to include information about the current operating conditions (the voltage and temperature) that are then used when making decision whether to accept or reject a fingerprint. For example, we may decide to accept a fingerprint with self-similarity index of 80%, providing that the environmental conditions during authentication are significantly different from those during enrollment.

6.4 Sub-segment Fingerprints

The experiments so far assumed that the state of an entire flash segment is used as a logical device fingerprint. Figure 9 indicates that the state bits after a partial erase appear to be uniformly randomly distributed, i.e., there is no spatial deviations within a segment. Consequently, we consider using n consecutive bits within a flash segment as a device fingerprint.

A series of experiments is conducted using fingerprints of n = 2,048, 1,024, 512, and 256 bits. In this analysis, a single 4,096-bit segment's enrollment fingerprint is divided into 4,096/n enrollment fingerprints. Likewise, the segment's fingerprint derived during authentication is divided into 4,096/n logical device fingerprints and the inter- and self- similarity indices are evaluated. Figure 15 shows the inter- and self-similarity indices as a function of the fingerprint length. The

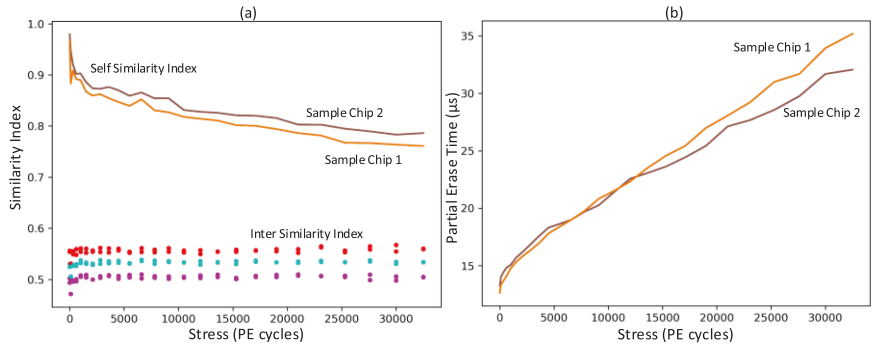

Fig. 16. Effect of PE cycles on Similarity Index (a) and partial erase time needed to generate a qualified fingerprint (b).

results show that even 256-bit fingerprints provide a healthy margin between the inter- and self-similarity indices without use of any encoding or compute intensive error correction techniques. This is true even for shorter fingerprints, but in the case of 128-bit and 64-bit fingerprints statistical outliers (marked in red and green dots) would overlap.

Assuming a 256-bit fingerprint is sufficient for identifying a logical device, the total number of logical devices considered in our experiment is 2,400 (three chips with 50 segments, each segment with 16 fingerprints). The number of bits needed for a unique identifier is thus $\lfloor \log_2 2400 \rfloor = 11.22$ bits, or 11.22/256=0.0438 bits of the identifier per a single flash bit. Using this metric, a 512-bit fingerprint can uniquely identify ~5.6 million of devices and a 1,024-bit fingerprint 31.37 trillion of devices.

6.5 Wear-out Considerations

One important concern is related to the robustness of the proposed fingerprint method in presence of segment aging. NOR flash memories can typically sustain 100,000 program-erase (PE) cycles before they may permanently fail. If we dedicate a flash segment to fingerprinting, then each authentication request results in one full PE cycle and one partial erase, so at least 50,000 authentication requests can be made before the segment is worn out if we assume that a single pass (steps 4–6 in Figure 11) is sufficient to generate AF. Figure 16(a) shows the self-similarity and intra-segment indices for two flash memory segments from two different chips as a function of the number of PE cycles. The authentication fingerprints derived after a certain number of PE cycles are compared to their enrollment counterparts that are retrieved before the segment is stressed. The stressing changes characteristics of the flash memory cells and has a negative impact on the self-similarity indices in both chips. The self-similarity indices drop somewhat sharply for the first 500 PE cycles, but further stresses result in mild degradation. Still, even after 30,000 PE cycles the self-similarity index is ~0.8. For the two sample chips presented in Figure 16(a), we use a linear fit to model the self-similarity index as a function of the number of PE cycles. The modeled changes of the self-similarity index are described as follows:

$$Self_{SI_{Stress}} = Self_{SI_0} - (k * 10^{-6} * Stress), \tag{2}$$

where *Stress* is the number of PE cycles, and k is the parameter capturing degradation of the similarity index relative to the index of the fresh memory block, $Self\_SI_0$. We find that the parameter k ranges between 4 and 5. Thus, $k = 5$ can be used to assess the minimum acceptable value of the self-similarity index in presence of stress.

The impact of stressing on inter-similarity, however, index is modest. The upper tail of the inter-similarity index distribution is below 0.57, leaving enough margin between the self-similarity and the inter-similarity indices. This confirms the robustness of the proposed fingerprint in presence of aging.

Another important aspect of aging is that it impacts the partial erase time needed to create a qualified fingerprint. Figure 16(b) shows the partial erase time needed to produce a qualified fingerprint as a function of the stress level. The partial erase time steadily increases with an increase in the stress level. However, the proposed algorithm for deriving fingerprints (see Figure 11) already includes provisions for modulating the partial erase time until a qualifying fingerprint is found. The only possible downside is that multiple attempts for finding authentication fingerprint may be needed.

6.6 Performance Considerations

Ideally, generating fingerprints should take as little time as possible. The proposed algorithm achieves a very good performance and can be implemented on a low-end microcontroller requiring as little as 4 KB of RAM memory. The majority of time is spent in full segment erase that takes ~25 ms on the microcontroller used in this study. In addition, the full segment programming (all bits are set to 0) takes ~3 ms, the partial erase less than 50 μs, and reading the state bits ~50 to 150 μs, depending on the number of reads (parameter N_R in Figure 11). In total, deriving a qualifying fingerprint on a first try takes ~30 ms. If multiple tries are needed (e.g., fingerprint fails a qualifying test because the erase time has shifted due to aging), then the time increases in multiples of ~30 ms. In our experiments, we were able typically to find a qualified fingerprint on the first try, but even with multiple tries the achieved performance compares favorably to other techniques that may require seconds and even minutes to derive a fingerprint [28]. The proposed method does not require any helper data [16] or helper functions [2].

Table 1 gives a summary of fingerprinting techniques in flash memories. It reports the type of flash memory used, throughput achieved in generating fingerprints/PUFs, size of the fingerprint in bits, and the memory overhead required by the extraction algorithm. The overhead is reported in the number of bits needed to extract one fingerprint bit. The throughput reported for our proposal is $136/n$ Kbits/s, where n indicates the number of retries. However, in our experiments $n = 1$ was sufficient in generating the authentication fingerprint. Thus, the best-case latency for generating and reading the fingerprint is 136 Kbits/s. The extraction algorithm requires one bit of memory per one bit of generated fingerprint, thus significantly outperforming other algorithms. The size of the fingerprint, memory overhead, and throughput achieved make the proposed method suitable for resource-constrained low-end microcontrollers.

7 RELATED WORK

Storing device identifiers and/or cryptographic keys in non-volatile memories, programmable fuses, or battery-backed RAMs are straightforward but costly approaches to device identification and authentication. However, the key issue with these approaches is that they can be easily cloned by motivated adversaries. Alternative approaches rely on deriving physical unclonable functions and fingerprints that are unique to each device, and difficult or impossible to clone.

Efforts to extract device PUFs and fingerprints have gained a lot of attention from the research community in the past two decades. These efforts focus on extracting physical properties of a device or its component that are unique to the device. One of the first PUFs known under the name physical one-way function is proposed by Pappu et al. [18]. It relies on non-integrated optical system with randomly distributed light scatters inside. Although it proved robust and resilient to modeling and cloning attacks, it requires an optical precision mechanism with exact positioning

Table 1. Summary of Fingerprinting Methods in flash Memories, Their Throughputs, and Memory Overheads

Method	Flash Type	Throughput	Evaluation: fingerprint size [bits], memory overhead per bit of fingerprint, latency, and implementation cost
Applies multiple partial programs to extract threshold variations in flash memory cells [28]	NAND Flash Memory	Not reported	*Size:* 16 Kbits *Overhead:* 1+size of (unit) *Latency:* High, requires repeated partial program operations
Applies multiple partial erases, partial program, and program disturbances pulses to induce disturbances [11]	NAND Flash Memory	1364.1 bits/s	*Size:* 128 bits, *Overhead:* ~142 (Bit-map method), ~16 (Position-map method) *Latency:* High, requires repeated flash operations
Lowers supply voltage to suspend erase operation and expose threshold voltage variation [2]	1.5T Superflash Memory	Not reported	*Size:* 512 Kbits *Overhead:* 2 *Latency*: Small *Special:* Requires lowering power supply
Repeatedly applies program operations with selected data to achieve program disturb [22]	NAND Flash Memory	9.09 Kbits/s	*Size:* 20 Kbits *Overhead:* ~1 *Latency:* High, requires repeated program operations
Applies repeated partial program operations to induce disturbances [16]	NOR Flash Memory	Not reported	*Size:* 3n bits *Overhead:* 8 *Latency:* High, requires repeated program operations *Special:* Requires BCH encoding
Applies repeated erase suspend operation for fingerprint generation [17]	Superflash Memory	266 bits/s	*Size:* 16 bits *Overhead*: 256 *Latency*: High, requires repeated partial erase operations
This work: applies simple partial erase operation to expose threshold voltage variation	Embedded NOR Flash Memory	136/n Kbits/s	*Size*: 4 Kbits *Overhead*: 1 *Latency*: Small

of the laser beam and CCD camera for readouts. In addition, it does not render itself suitable for integration and miniaturization that are desirable in integrated circuits.

Gassend et al. in their seminal work introduced silicon random functions that can be used to identify and authenticate individual integrated circuits [4]. They propose several implementations based on a parameterized self-oscillating circuit with a delay block whose transient response depends on manufacturing process variations unique to each IC. They demonstrate that the proposed circuits implemented on Field Programmable Gate Arrays are robust in presence of significant environmental variations. Lee et al. introduced arbiter-based PUF circuits [14] that use delay variations in symmetric racing paths through a series of switching elements. Suh and Devadas introduce ring oscillator-based PUF in Reference [24]. Named RO PUF, this circuit does not depend on a multiplexer or an arbiter, but on delay loops and counters, thereby paving way for an easy implementation in FPGA and ASICs. All these proposed circuits can be utilized for identification, authentication, and key generation. However, they require custom circuits solely dedicated to this

function to be added to ICs, thus increasing their cost. In addition, they do not solve the problem of identification and authentication of the existing ICs.

To address the cost concerns associated with prior proposals, many research groups have turned their attention to using the existing system components for identification and key-generation. A number of techniques relies on SRAM, DRAM, and non-volatile flash memories to extract device fingerprints.

Holcomb et al. [8] propose using the state of power-up SRAM memory cells to create device fingerprints. The stable state that an SRAM cell transitions to after power up depends on manufacturing variations and thus can uniquely identify an SRAM chip. This solution has been demonstrated to be reliable in both stand-alone SRAM chips and embedded SRAM blocks. However, it requires supervisory access to the SRAM memory as it relies on powering-up SRAMs to generate fingerprints. Closely related to this work is a method that utilizes SRAM power-up states to generate a secret key to an algorithmic one-way hash function that is used for IP protection in FPGAs [5]. Bacha and Teodorescu utilize SRAMs in high-end processor cache memories to identify devices [1]. This technique is based on lowering the supply voltage to the cache memory to induce errors. The distribution of errors in cache lines is used to authenticate the chip.

A number of research proposals for DRAM-based PUFs that utilize properties of DDR3 and DDR4 memories has recently emerged [6, 12, 13, 20, 22, 24–26] These techniques use different properties of DRAM memories to extract unique identifiers, such as: (a) write-failures where write-duty cycles are selected to induce failure in write operations [6]; (b) access-latency variations of individual cells based on their location [13]; state of DRAM cells after power-up [27]; refresh-pausing [25]; read latency variation to induce errors [12]; and decay based on disable/postpone refresh cycles to induce decays [23]. Whereas these solutions are of great interest in computer platforms with DRAM chips, they are not useful in embedded platforms that do not include DRAM modules.

Flash memories, especially high-density NAND flash memories, have been utilized for creating device fingerprints. Prabhu et al. [20] summarize seven ways of extracting fingerprints from NAND flash memories. These techniques are based on program disturb, read disturb, per bit program operation latency variation (single page and multi-page), erase latency, read latency, whole page program latency, and susceptibility of cells to program/erase induced wear. The authors show that the program disturb and the program operation latency variation give quality PUFs that can be generated relatively quickly, whereas the read disturb is also adequate but very slow. Other techniques are deemed inadequate. Wang et al. [28] propose a NAND-based solution that utilizes repeated partial program operations to generate fingerprint as well as true random numbers. The number of partial program operations needed to turn each cell into the programmed state is used to create a unique device fingerprint. Jia et al. [11] propose three techniques for PUF-based key generator using NAND flash memory, namely, partial erasure, partial programming, and program disturbance. These techniques are suitable for high-density NAND flash memories, require significant storage and compute resources, and are relatively slow, thus not directly applicable to embedded systems. Sakib et al. [22] propose an aging-resistant PUF for NAND flash memory utilizing a program disturb method to extract inherent process variations unique to each chip. To counter aging effect, the authors propose a tunable algorithm for extraction of authentication PUFs, ensuring high accuracy for 1,000 authentication requests.

Mandadi [16] proposes a PUF generation in standalone NOR flash memories that utilizes partial program operations of individual memory locations, using a memory address as a request. The position of a flash bit that flips first is recorded and encoded as a 3-bit PUF response (in case of byte locations). Responses from a range of consecutive memory locations are stitched together to create a PUF. One drawback of this approach is that it may require multiple partial

program operations if several flash memory bits flip concurrently. Alternately, multiple flash bit positions can be used depending on the length of PUF response desired. However, the computational and storage requirements of this approach may also exceed capability of low-end microcontrollers. Clark et al. propose a fingerprinting scheme in 1.5-T standalone NOR Superflash memories using partial erase operation [2]. The scheme relies on partially erasing flash memory at a reduced voltage supply; if the percentage of the erased cells reaches ≥ 50%, then it represents a fingerprint.

8 CONCLUSIONS

This article introduces a new technique for generating device fingerprints that exploits properties of partially erased NOR flash memory cells. In-system programmable NOR flash memories, a standard part in all modern microcontrollers, are typically used for storing firmware and read-only data, but they can be erased and programmed during normal system operation. The proposed technique utilizes the state of partially erased flash memory segments to reveal unique physical properties of flash memory cells through a standard digital interface. These physical properties are artifacts of semiconductor process variations that are unique for each flash memory segment and each physical device. We provide an in-depth characterization of NOR flash memory behavior when using partial erase operations and introduce the algorithms for generating the enrollment and authentication fingerprints.

The proposed technique is thoroughly evaluated to prove that it produces fingerprints that are unique, repeatable, and robust in presence of changes in ambient temperature and variations in operating voltage. In addition, we show that the proposed method can tolerate effects of aging.

The proposed technique offers a number of advantages over the existing approaches: (a) it requires no additional hardware support, (b) it is solely implemented in software and can be easily implemented in modern microcontrollers, (c) it produces unique robust fingerprints that are resilient to changes in the operating conditions, (d) it has good performance, and (e) the algorithm self-adapts to changes due to flash memory aging effects.

REFERENCES

[1] Anys Bacha and Radu Teodorescu. 2015. Authenticache: Harnessing cache ECC for system authentication. In *Proceedings of the 48th International Symposium on Microarchitecture (MICRO'15)*. Association for Computing Machinery, 128–140. DOI: https://doi.org/10.1145/2830772.2830814

[2] Lawrence T. Clark, James Adams, and Keith E. Holbert. 2019. Reliable techniques for integrated circuit identification and true random number generation using 1.5-transistor flash memory. *Integration* 65, (2019), 263–272. DOI: https://doi.org/10.1016/j.vlsi.2017.10.001

[3] Adam R. Duncan, Matthew J. Gadlage, Austin H. Roach, and Matthew J. Kay. 2016. Characterizing radiation and stress-induced degradation in an embedded split-gate NOR flash memory. *IEEE Trans. Nucl. Sci.* 63, 2 (2016), 1276–1283. DOI: https://doi.org/10.1109/TNS.2016.2540803

[4] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. 2002. Silicon physical random functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS'02)*. Association for Computing Machinery, 148–160. DOI: https://doi.org/10.1145/586110.586132

[5] Jorge Guajardo, Sandeep S. Kumar, Geert-Jan Schrijen, and Pim Tuyls. 2007. FPGA intrinsic PUFs and their use for IP protection. In *Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems (CHES'07)*. Springer-Verlag, 63–80. DOI: https://doi.org/10.1007/978-3-540-74735-2_5

[6] Maryam S. Hashemian, Bhanu Singh, Francis Wolff, Daniel Weyer, Steve Clay, and Christos Papachristou. 2015. A robust authentication methodology using physically unclonable functions in DRAM arrays. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE'15)*, 647–652. DOI: https://doi.org/10.7873/DATE.2015.0308

[7] Charles Herder, Meng-Day Yu, Farinaz Koushanfar, and Srinivas Devadas. 2014. Physical unclonable functions and applications: A tutorial. *Proc. IEEE* 102, 8 (2014), 1126–1141. DOI: https://doi.org/10.1109/JPROC.2014.2320516

[8] Daniel E. Holcomb, Wayne P. Burleson, and Kevin Fu. 2009. Power-Up SRAM state as an identifying fingerprint and source of true random numbers. *IEEE Trans. Comput* 58, 9 (2009), 1198–1210. DOI: https://doi.org/10.1109/TC.2008.212

[9] Texas Instruments. 2008. MSP430 Flash memory characteristics. Retrieved from http://www.ti.com/lit/an/slaa334b/slaa334b.pdf.

[10] Texas Instruments. 2009. MSP430F543x, MSP430F541x Mixed-Signal Microcontrollers datasheet (Rev. F). Retrieved from http://www.ti.com/lit/ds/symlink/msp430f5438.pdf.

[11] Shijie Jia, Luning Xia, Zhan Wang, Jingqiang Lin, Guozhu Zhang, and Yafei Ji. 2015. Extracting robust keys from NAND flash physical unclonable functions. In *Information Security* (Lecture Notes in Computer Science), Springer, Cham, 437–454. DOI: https://doi.org/10.1007/978-3-319-23318524

[12] Jeremie S. Kim, Minesh Patel, Hasan Hassan, and Onur Mutlu. 2018. The DRAM latency PUF: Quickly evaluating physical unclonable functions by exploiting the latency-reliability tradeoff in modern commodity DRAM devices. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'18)*, 194–207. DOI: https://doi.org/10.1109/HPCA.2018.00026

[13] Donghyuk Lee, Samira Khan, Lavanya Subramanian, Saugata Ghose, Rachata Ausavarungnirun, Gennady Pekhimenko, Vivek Seshadri, and Onur Mutlu. 2017. Design-induced latency variation in modern DRAM chips: Characterization, analysis, and latency reduction mechanisms. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'17 Abstracts)*. Association for Computing Machinery, 54. DOI: https://doi.org/10.1145/3078505.3078533

[14] J. W. Lee, Daihyun Lim, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas. 2004. A technique to build a secret key in integrated circuits for identification and authentication applications. In *Proceedings of the Symposium on VLSI Circuits*. 176–179. DOI: https://doi.org/10.1109/VLSIC.2004.1346548

[15] Muqing Liu, Chen Zhou, Qianying Tang, Keshab K. Parhi, and Chris H. Kim. 2017. A data remanence-based approach to generate 100% stable keys from an SRAM physical unclonable function. In *Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED'17)*, 1–6. DOI: https://doi.org/10.1109/ISLPED.2017.8009192

[16] Harsha Mandadi. 2017. Remote integrity checking using multiple PUF-based component identifiers. Retrieved from https://vtechworks.lib.vt.edu/handle/10919/78200.

[17] The-Nghia Nguyen, Sunghyun Park, and Donghwa Shin. 2020. Extraction of device fingerprints using built-in erase-suspend operation of flash memory devices. *IEEE Access* 8, (2020), 98637–98646. DOI: https://doi.org/10.1109/ACCESS.2020.2995891

[18] Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. 2002. Physical one-way functions. *Science* 297, 5589 (2002), 2026–2030. DOI: https://doi.org/10.1126/science.1074376

[19] Prawar Poudel, Biswajit Ray, and Aleksandar Milenkovic. 2019. Microcontroller TRNGs using perturbed states of NOR flash memory cells. *IEEE Trans. Comput.* 68, 2 (2019), 307–313. DOI: https://doi.org/10.1109/TC.2018.2866459

[20] Pravin Prabhu, Ameen Akel, Laura M. Grupp, Wing-Kei S. Yu, G. Edward Suh, Edwin Kan, and Steven Swanson. 2011. Extracting device fingerprints from flash memory by exploiting physical variations. In *Trust and Trustworthy Computing* (Lecture Notes in Computer Science), Springer, Berlin, 188–201. DOI: https://doi.org/10.1007/978-3-642-21599-5_14

[21] Sami Rosenblatt, Srivatsan Chellappa, Alberto Cestero, Norman Robson, Toshiaki Kirihata, and Subramanian S. Iyer. 2013. A self-authenticating chip architecture using an intrinsic fingerprint of embedded DRAM. *IEEE J. Solid-State Circuits* 48, 11 (2013), 2934–2943. DOI: https://doi.org/10.1109/JSSC.2013.2282114

[22] Sadman Sakib, Aleksandar Milenković, Md Tauhidur Rahman, and Biswajit Ray. 2020. An aging-resistant NAND flash memory physical unclonable function. *IEEE Trans. Electron Devices* 67, 3 (2020), 937–943. DOI: https://doi.org/10.1109/TED.2020.2968272

[23] André Schaller, Wenjie Xiong, Nikolaos Athanasios Anagnostopoulos, Muhammad Umair Saleem, Sebastian Gabmeyer, Boris Škorić, Stefan Katzenbeisser, and Jakub Szefer. 2019. Decay-based DRAM PUFs in commodity devices. *IEEE Trans. Dependable Secure Comput.* 16, 3 (2019), 462–475. DOI: https://doi.org/10.1109/TDSC.2018.2822298

[24] G. Edward Suh and Srinivas Devadas. 2007. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th Annual Design Automation Conference (DAC'07)*. Association for Computing Machinery, 9–14. DOI: https://doi.org/10.1145/1278480.1278484

[25] Soubhagya Sutar, Arnab Raha, and Vijay Raghunathan. 2016. D-PUF: An intrinsically reconfigurable DRAM PUF for device authentication in embedded systems. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES'16)*. Association for Computing Machinery, 1–10. DOI: https://doi.org/10.1145/2968455.2968519

[26] B. M. S. Bahar Talukder, Biswajit Ray, Mark Tehranipoor, Domenic Forte, and Md Tauhidur Rahman. 2018. LDPUF: Exploiting DRAM latency variations to generate robust device signatures. Retrieved from http://arxiv.org/abs/1808.02584.

[27] Fatemeh Tehranipoor, Nima Karimian, Wei Yan, and John A. Chandy. 2017. DRAM-based intrinsic physically unclonable functions for system-level security and authentication. *IEEE Trans. Very Large Scale Integr. Syst.* 25, 3 (2017), 1085–1097. DOI: https://doi.org/10.1109/TVLSI.2016.2606658

[28] Yinglei Wang, Wing-kei Yu, Shuo Wu, Greg Malysa, G. Edward Suh, and Edwin C. Kan. 2012. Flash memory for ubiquitous hardware security functions: True random number generation and device fingerprints. In *Proceedings of the IEEE Symposium on Security and Privacy*. 33–47. DOI: https://doi.org/10.1109/SP.2012.12

Received April 2020; revised December 2020; accepted December 2020