Memory-Based PUFs Are Vulnerable as Well: A Non-Invasive Attack Against SRAM PUFs

B. M. S. Bahar Talukder, *Graduate Student Member, IEEE*, Farah Ferdaus, *Graduate Student Member, IEEE*, and Md Tauhidur Rahman, *Senior Member, IEEE*

***Abstract*— Memory-based physical unclonable functions (mPUFs) are widely accepted as highly secure because of the unclonable and immutable nature of manufacturer process variations. Although numerous successful attacks have been proposed against PUFs, mPUFs are resistant to non-invasive attacks as the mPUF does not support the open-access protocol. Hence, existing attacks against mPUFs mostly rely on invasive/semi-invasive techniques or at least require physical access to the target device, which is not always feasible. In this paper, we experimentally demonstrate that signatures generated from two memory chips may have highly correlated properties if they possess the same set of specifications and a similar manufacturing facility, which is used to mount a non-invasive attack against memory-based PUFs. Our proposed technique shows that if an attacker has access to a device similar to the victim's one, the attacker might be able to guess up to ∼45% of the challenge-response pairs of a 64-bit SRAM PUF.**

***Index Terms*— PUF, PUF attack, PUF modeling, non-invasive attacks on weak PUF.**

I. Introduction

IN THE Internet-of-Things (IoT) era, most electronic devices are connected to other devices through the internet or local network, making electronic devices more vulnerable to many active and passive attacks. The conventional cryptographic and authentication techniques overcome such security issues to some extent; however, they mostly rely on some secret key and assume that the device is secured as long as the key remains secret. Most electronic devices store their secret keys in a non-volatile memory such as electrically erasable programmable read-only memory (EEPROM) in such cryptographic and authentication techniques. Unfortunately, in practice, maintaining the key's secrecy in non-volatile memory is not always feasible and is often unveiled by different active or passive attacks. Researchers came up with an intriguing solution to solve such a problem: Physical Unclonable Function (PUF). PUF generates a reproducible and unclonable key on the fly and eliminates the necessity to store the key; hence, it prevents the attacker from stealing the key. More precisely, a PUF is a circuit that can generate a unique "response" based on the "challenge" (or "stimuli") provided by the external source. The PUF response's uniqueness appears from the inherent and unavoidable random manufacturing process variations [1], [2]. Process variation is an intrinsic phenomenon that causes uncontrollable variation in integrated circuits' electronic properties [3].

Although traditional PUFs have desirable security features, it is still not a viable solution for small and low-cost hardware as it requires additional circuitry. To avoid this problem, researchers proposed memory-based PUFs (mPUFs) [1], [2], [4], [5] that do not require any additional circuitry, as most of the electronic devices are equipped with memory for their regular operation. At present, mPUFs have been successfully deployed in many commercial devices, including FPGA, microcontroller, and application-specific ICs (ASIC) [6]–[11]. Memory-based PUFs are classified as weak PUFs because they have a limited number of challenge-response pairs (CRPs). In contrast, the traditional PUFs such as arbiter PUF are known as strong PUF because they have a large number of CRPs [12]–[15]. As the number of CRPs is limited for weak PUF, an attacker can easily clone all CRPs if the attacker gains access, which is not possible for strong PUFs. Hence, weak PUFs' CRPs are access-restricted as opposed to strong PUF [14], [15].

There have been several invasive and non-invasive techniques to predict/model PUF responses, although PUFs are widely accepted as highly secure because of the unclonable and immutable nature of manufacturer process variations [16]–[20]. These attacks can be categorized into (i) *protocol attacks* and (ii) *silicon attacks*. In the *protocol attack*, the attackers usually exploit existing design flaws in the PUF network protocol such as gaining temporary access on a PUF, re-using the PUF responses from previous sessions, creating a covert channel to perform malicious activity on PUF, leaking information via error correction scheme, etc. [16], [17]. To defend against such attacks, we need to fix the network protocol weaknesses and vulnerabilities as soon as they are exposed.

On the other hand, in the *silicon attack*, an attacker tries to (i) imitate the characteristics of the PUF device and generate the correct responses, (ii) forge/clone all possible CRPs, and (iii) modify the PUF responses. For the first case, the attacker needs to derive some deterministic relationship between the challenges and responses using known

Manuscript received February 20, 2021; revised May 19, 2021; accepted July 8, 2021. Date of publication July 30, 2021; date of current version August 19, 2021. This work was supported in part by the National Science Foundation under Grant CNS-1850241. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Ulrich Rührmair. *(Corresponding author: B. M. S. Bahar Talukder.)*

The authors are with the Department of Electrical and Computer Engineering, Florida International University, Miami, FL 33174 USA (e-mail: bbaha007@fiu.edu; fferd006@fiu.edu; mdtrahma@fiu.edu).

Digital Object Identifier 10.1109/TIFS.2021.3101045

challenge-response pairs (CRPs) [21]. However, knowing CRPs requires an open-access interface which is only possible for strong PUFs [14], [15]. On the other hand, forging/cloning or modifying mPUF output requires physical access and might involve invasive/semi-invasive attack methodologies that are impractical to mount in a protected environment [22]–[26].

In contrast, our proposed technique does not require access to the target mPUFs. In this work, we have experimentally demonstrated that if two mPUFs share the same manufacturing resources (similar specification and fabrication facility), they may have significantly correlated properties in their signatures. We show that an attacker can easily learn signature heuristics if he has access to similar mPUFs as the victim's one. Afterward, the attacker can use this information to determine the challenge-response pair of the target mPUF. The major contributions of this paper include:

- We briefly discuss the weakness in PUF metrics, which are widely used to quantify the "goodness" of many mPUFs. Without proper attention, an attacker may use such weaknesses to attack mPUFs.
- We show that two memory chips sharing the same manufacturing resources (i.e., specification, manufacturer, fabrication plant, etc.) generate correlated signature properties.
- We demonstrate that if an attacker has access to similar devices to the victim's mPUF device, the attacker can learn the characteristics of the target mPUF and perform a heuristic attack to recover mPUF responses.
- We propose a set of recommendations to secure mPUF.

II. Background

A. General PUF Protocol

PUF follows almost a similar cryptographic protocol in most security applications, as shown in Fig. 1 [27]–[29]. PUF operation consists of two phases. In the enrollment/registration/key generation phase, the PUF device generates a response R_{C^i} depending on a given challenge C_i. Then, an ECC encoder computes the corresponding error correction code/helper data h_{C^i} and generates the key, $K(R_{C^i})$ (e.g., using hash). In the key re-generation or authentication phase, the PUF circuits again receive the same challenge C_i, and generates the corresponding response R'_{C^i}. Due to the different internal and external noises (variations in operating condition, device aging, etc.), some errors are always expected in the generated response R'_{C^i} (compared to R_{C^i}). However, corrected response R''_{C^i} is computed using the helper data h_{C^i}. Subsequently, the key, $K(R''_{C^i})$, is generated from the R''_{C^i}. If the PUF circuit is not altered in between these two phases, the $K(R_{C^i})$ and $K(R''_{C^i})$ should be the same.

B. Fuzzy Extractor

The Fuzzy extractor is used to enhance the security of the PUF-based key [30]–[34]. The security enhancement is performed by (i) removing noise without leaking significant information and (ii) extracting the entropy (i.e., increase the entropy density on fuzzy output) [30]. It has

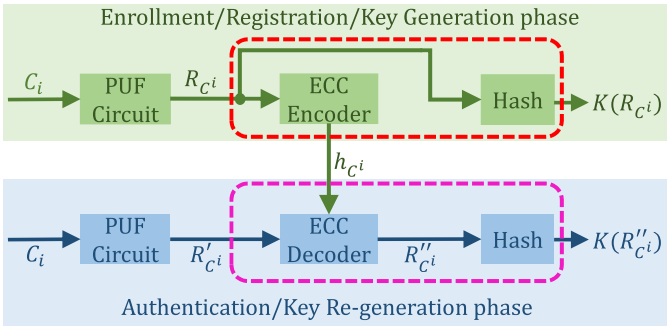


Fig. 1. PUF protocol.

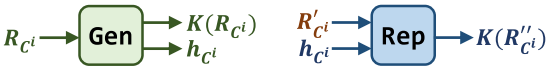


Fig. 2. Basic blocks of fuzzy extractor.

two major blocks: (i) a generate algorithm (`Gen`) in the enrollment/registration phase and (ii) a reproduce algorithm (`Rep`) in the re-generation/authentication phase (Fig. 2). The `Gen` and `Rep` replace the components from the red and magenta boxes of Fig. 1, respectively. The `Gen` takes the raw PUF response R_{C^i} and produces the output key $K(R_{C^i})$ and h_{C^i}. In the authentication/re-generation phase, the `Rep` takes the noisy input R'_{C^i} and h_{C^i} and produces the corrected key $K(R''_{C^i})$.

The noisy behavior of PUFs requires to allow some errors, which reduce the entropy of the input PUF key (R). However, this problem can be avoided, if the output key length ($K(R)$) does not exceed the fuzzy min-entropy [30], [31]. The fuzzy min-entropy (H_∞^{fuzzy}) is defined as follows-

$$H_\infty^{fuzzy}(\mathcal{R}) = -\log \max_{R'} \{Pr[\mathcal{R} \in B_q(R')]\} \quad (1)$$

Here, $\mathcal{R}$ is the distribution of the original response R. To maximize the chance; the attacker would select R' that maximize the total probability mass of all responses (drawn from distribution $\mathcal{R}$) within the ball $B_q(R')$ of radius q around R' [30], [31]. The main difficulty of designing the fuzzy extractor is to estimate $Pr[\mathcal{R} \in B_q(R')]$. For a reasonable estimation of the probability distribution, one might require a large number of sample sources. Moreover, an attacker can always make a better probabilistic estimation with a larger set of samples [30], [31].

Although Fuzzy extractor is widely used in many bio-metric authentications and high-end PUF authentication, a low-cost error correction scheme may replace it in many low-cost PUF-based devices [1], [28], [29], [34]–[38], where the error correction scheme might not maintain the constraint presented by Eq.1. In a most simple design, the error corrector does not extract the entropy or increase the entropy density at the output; instead, it only corrects the error of the inputs. Many researchers proposed such error correctors to replace the fuzzy extractor in low-cost applications [1], [28], [29], [34], [38]. In this paper, we assume a simple error correction scheme (i.e., the entropy of R remains unaffected) instead of

a complex fuzzy extractor and explore the impact of both the error correction scheme and the non-uniform distribution of the PUF response. However, in Sec. VI-D, we also discuss the scope and limitation of our proposed attack by considering a fuzzy extractor.

C. Security

The security of PUF keys is dependent upon two features of the key: uniqueness and uniformity [39], [40]. The first property indicates that the PUF response should be unique and should not collide if multiple PUF devices are stimulated with the same challenge. In other words, if the responses generated from two PUFs (with the same challenge) are properly random, the *Hamming distance*[1] between two PUF responses should be ∼50%. However, if the number of test devices is large, then one-to-one *Hamming distance* becomes a very expensive metric to test (due to a large number of possible combinations). Furthermore, mPUFs with large address-space might be capable of producing multiple challenge-response pairs (CRP) using different address locations as the challenge [41]. Consequently, assessing *Hamming distance* for each CRP makes it computationally expensive. Fortunately, *bit-aliasing* is a cheaper alternative to test uniqueness. For this metric, we create a binary string using all memory-bits from a specific bit-location of all PUF devices and measure the *Hamming weight*[2] [39], [40] (Eq. 2).

$$\beta_{l,C} = \frac{1}{k}\sum_{i=1}^{k} b^i_{l,C} \tag{2}$$

Here, $\beta_{l,C}$ is the bit-aliasing computed at the lth bit using k PUF devices with a unique challenge C, and $b^i_{l,C}$ is the lth bit response generated from the ith PUF device. Hence, for the n-bit responses generated from all k PUF devices, we have a set of β. To make it simpler, researchers only focus on the average *bit-aliasing* defined by the following equation.

$$\beta_C = \frac{1}{n}\sum_{l=1}^{n} \beta_{C,l} \tag{3}$$

For a set of ideal PUFs, β_C (with unique challenge C) should be close to 0.5. This condition suggests that the probability of getting "0" (or "1") from a specific bit location over multiple PUFs is exactly 50%.

Besides being unique, a PUF response also needs to be uniform, i.,e., the PUF response should have a uniform distribution of "0" and "1". To satisfy this condition, the average *Hamming weight* of the PUF responses should be close to 50% [39], [40]. However, in Sec. III-C, we demonstrated that such a uniformity test based on *Hamming weight* is insufficient and opens space for our proposed attacks.

[1]*Hamming distance* (or simply "distance") is defined as the number of bit difference between two binary strings.

[2]*Hamming weight* is defined as the total number "1" (or "0") in a bit-stream.

III. Vulnerabilities of mPUFs and Weakness of PUF Metrics

In this section, we revisit several aspects of mPUFs overlooked by researchers. From now on, we focus on SRAM PUFs (the most common type of mPUF).

A. Source of PUF Randomness

Traditionally, random process variations (PVs) have been considered the primary source of randomness, creating a PUF unique and secure. However, the signatures generated from two memory chips may have highly correlated properties due to architectural variation and die-to-die variation, which will be used as side-channel information for non-invasive attacks against mPUFs. We investigate and characterize the following important factors that result in these deterministic properties.

- *Architectural and Layout Similarities:* The similarity in architecture and layout of SRAM memory chips are among the main reasons for correlated signature properties between two mPUFs with the same model number (i.e., part number). All semiconductor memory chips share a very similar row-column architecture [42], [43]. However, the address decoder circuitry for logical to physical address mappings, control circuitry to determine the I/O state (read/write), optimal array structure to the trade-off between cost and performance, power-saving techniques, the structure of redundant blocks (rows or column) to replace corrupted memory arrays entirely or partially, etc. vary from one manufacturer to another or from one model to another. Moreover, manufacturers might also shrink the die size to reduce the cost per cell, causing the memory chips to be more susceptible to noise and less robust. Additionally, the manufacturer may follow different SRAM cell structures (e.g., 4T, 5T, 6T, 7T, 8T, 9T) for further optimization [42], [43]. Some of those architectures are asymmetric and are more likely to produce biased outputs. Although symmetric SRAM architectures are the preferred choice for SRAM PUFs [44], a symmetric SRAM cell may still produce deterministic biased outputs because of asymmetric fabrication technology (e.g., asymmetric silicon doping [45]), asymmetric driving strength of bitline pair [46], etc. Furthermore, a small mismatch in coupled inverters' layout (e.g., a slight difference in routing) might also induce slight asymmetricity in SRAM cells [47]. Although such small asymmetricity might not impact the regular operation of SRAMs, that might still be visible from the SRAM start-up data statistics (by slightly favoring logic "0" or "1").
- *Process Variations (PVs):* PV is an intrinsic phenomenon during the fabrication process that introduces the variability on transistors' attributes, interconnecting metal, and dielectric layers. The PV can be divided into two categories-global PV and local PV [48]. However, the PV can be broken down into finer grains, as shown in Fig. 3 [48]–[52]. The systematic variations are deterministic and correlated with the locality of the chip layout, characteristics of the silicon wafer/fabrication facility, node technology, etc. [53]. In contrast, the random PV is entirely indeterministic. Furthermore, the systematic PV can vary from wafer to

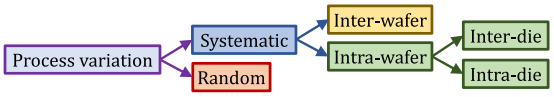


Fig. 3. Different types of process variation [48]–[52].

wafer (i.e., inter-wafer) and within the wafer (intra-wafer). The inter-wafer PV impacts each silicon wafer differently and may depend on the wafer characteristics. Moreover, the intra-wafer variation describes the variation among dies fabricated in a single wafer. The intra-wafer process variation may be further divided into two groups- inter-die variation and intra-die variation. The inter-die variation affects each dies differently (i.e., multiple dies fabricated in the same wafer). On the other hand, the intra-die variation defines the spatially correlated variation within a single die.

The random process variation is extremely difficult to model, especially to determine the exact mismatch between coupled inverters. The inter-wafer process variation might also be challenging to model as it depends on the wafer manufacturing process and corresponding intrinsic parameters. However, the intra-wafer variation might have deterministic patterns and might be learned by statistical analysis [48], [54], [55].

Additionally, in some process technology, the fabrication plant may have a specific set of design rules depending on the layout pattern (for example, different design rules for a mirrored layout pattern). The manufacturer might need to break the symmetricity of the layout to facilitate such design rules. Hence, a perfectly symmetric 6T-SRAM cell layout can become asymmetric after fabrication. Recent studies show that systematic process variation can significantly impact the SRAM start-up data (i.e., might get biased to '0' or '1') [56].

B. Acceptance of Errors in PUF Protocol

If a q-bit error is allowed in an n-bit response (R), then all n-bit binary strings at a q-bit distance (from R) will be accepted in the authentication/key re-generation phase (Fig. 1). Now, using the *covering code* theory [57], [58], an attacker can reduce the number of possible patterns that cover all possible n-bit strings (i.e., smaller search space to find the valid key). For example, if $n = 16$ and $q = 2$, then only <960 patterns are sufficient to represent all 2^{16} patterns [58]. Hence, only allowing 2-bit error will reduce the entropy of response (R) from 16 bits to $< \log_2(960) < 10$ bits [59]. However, a properly designed fuzzy extractor can eliminate the PUF weakness introduced by the error correction scheme (see Sec. II-B).

C. Weakness of PUF Metrics

In general, to get a uniform response from an n-bit PUF, the probability of getting "0" or "1" at any bit location should be $\frac{1}{2}$ (denoted with $p = \frac{1}{2}$). Alternatively, all possible patterns encoded with n-bit should have an equal probability of being a PUF response. If a PUF satisfies this condition,

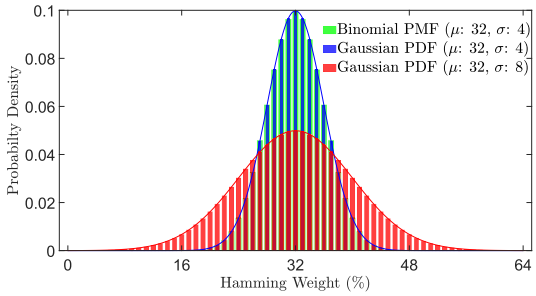


Fig. 4. Binomial vs. Gaussian distribution ($n = 64$).

the *Hamming weight* (*HW*) of all the possible responses should follow a Binomial distribution [60] (see Appendix A) with a mean ($\mu = np$) and standard deviation ($\sigma = \sqrt{np(1-p)}$). If the μ and σ are large enough, such a Binomial distribution can be approximated with a Gaussian distribution of the same mean and standard deviation (i.e., $\mu = np$ and $\sigma = \sqrt{np(1-p)}$) [61]. However, this condition is not checked adequately in many PUF related works [1], [2], [4], [5], [62], and sometimes, a Gaussian curve is fitted around the distribution of *Hamming weight* with an arbitrarily chosen σ (i.e., only satisfying the condition, $\mu = np$). If the condition of μ or σ is not appropriately maintained, the PUF might produce biased outputs (see Appendix B).

In Fig. 4, we present a comparison between the Binomial distribution and the Gaussian Distribution with $n = 64$. For the Binomial distribution, we assume that all patterns encoded with 64-bit Binomial string have an equal probability of occurrence (i.e., $p = 0.5$). This Binomial distribution returns a mean, $\mu = np = 32$ and standard deviation, $\sigma = \sqrt{np(1-p)} = 4$ (plotted with green bar). This Binomial distribution can be approximated with a Gaussian curve with the same mean and standard deviation (plotted with blue color). However, if $\sigma \neq \sqrt{np(1-p)}$ and fitted with an arbitrarily chosen σ (plotted in red color), a set of patterns with some specific *Hamming weights* have a higher chance to be the PUF outcome (see Appendix B). In such cases, it is much easier to reveal the key by the heuristic analysis of pattern frequency in such instances. Note that the perfect Binomial distribution of *Hamming weight* does not solely guarantee the perfect randomness of CRPs. For example, two different patterns, X_1 and X_2, may share the same *Hamming weight* of h_X; however, a perfect probabilistic value of h_X (drawn from the binomial distribution) does not imply that X_1 and X_2 have an equal probability of being PUF responses (see Appendix B). Therefore, we recommend checking the uniformity of pattern distribution on mPUF signature data (see Sec. VI).

IV. Threat Model

In most attacks on weak mPUFs, attackers require physical access to the target PUF device at least once (see Sec. I). However, gaining physical access to most devices is practically difficult. The non-invasive attack model proposed in this paper requires no physical access to the target PUF. We present our proposed threat model below.

1) The attacker needs to know the memory model (manufacturer and part number) of the target mPUF, which can be revealed by analyzing the victim's device information, such as the device manufacturer, model/variant, version, etc. They can mostly obtain that information by using existing *detection* techniques [63], [64] or/and from the victim's browser fingerprints.
2) The attacker can own other PUF devices with the same vendor and part number as the victim's device. We also assume that the attacker can apply any invasive/semi-invasive/non-invasive technique to characterize his acquired PUF devices and learn signature heuristics, which can be used to attack the victim's PUF.
3) The attacker also knows about the error margin in the key re-generation/authentication phase (Fig. 1). The previous assumption can relax this assumption. If the attacker has access to similar PUF models, he or she can estimate this error margin. For example, if a bunch of SRAM PUF of model "X" produces an average q-bit error between two valid operating conditions, the attacker can assume that the victim's device (of model "X") will also produce on an average q-bit error in those two operating conditions.
4) In an unsuccessful attempt made by the attacker, the degree of mismatch between $K(R_{C^i})$ and $K(R''_{C^i})$ (or between R_{C^i} and R''_{C^i}) is not released in the authentication/key re-generation phase (see Fig. 1).

The above assumptions are more practical than all previous attacking scenarios. These assumptions also provide an insight into the attack surface an adversary can exploit. Our proposed attack is non-invasive as we do not require any access to the victim's PUF. In Sec. VII, we will discuss a set of countermeasures that can narrow down the attack surface.

V. Proposed Method

To recover the signature/key from mPUFs, at first, we discover possible vulnerabilities of mPUF. Second, we propose an attacking scheme on mPUF by exploiting these mPUF vulnerabilities. Finally, we propose a set of techniques to generate secure Keys/signatures. We explain the first and second tasks in the following two subsections and the third task in Sec. VII.

A. Analyzing PUF Vulnerabilities

1) Limited Randomness: Our research suggests that DRAM fingerprints are not entirely random; instead, chips with the same model number can produce signatures with correlated and deterministic properties [65]. In that work, we used standalone DRAM modules with latency-based signatures [5]. We have also found a similar trait for SRAM chips. That is, SRAM chips of the same model number develop deterministic and correlated nature on their signatures. We investigate seven critical features to understand these deterministic properties of SRAM start-up data texture. Below, we provide a brief overview of these features.

- *Feature 1:* The start-up value of an SRAM cell is either '1' or '0'. The fraction of cells that exhibit logic '1' provides some information about the correlation (i.e., similarity or dissimilarity) between SRAM signatures.
- *Feature 2:* In our experiment, each of the memory chips has a 16-bit parallel interface, which means selecting an address provides a 16-bit word. In this feature, we quantify the standard deviation of the *Hamming weights* across all the 16-bit words.
- *Feature 3:* We divide each memory into 16 segments, where i^{th} segment contains i^{th} bit of all 16-bit words. Then the standard deviation of the *Hamming weights* of the segments is taken as feature 3.
- *Feature 4:* The compressibility (compression ratio) of start-up data (i.e., signature or fingerprint) is selected as a feature. This enables us to evaluate the randomness of start-up data.
- *Feature 5:* Each memory chip is divided into multiple segments containing memory cells from 512 consecutive words. The *Hamming weight* is computed from each segment, and the standard deviation is taken as the fifth feature. This feature provides information about the spatial locality in the start-up data.
- *Feature 6:* We quantify the noisy characteristics of start-up data. From multiple trials, the noisiest cells produce '1' or '0' with equal probability. We recorded the start-up data for each chip 20 times and counted the cells that flip 8 to 12 times of trials.
- *Feature 7:* Each 16-bit word has a *Hamming weight* within the range $[0, 16]$, which enables us to form a histogram of 17 bins. In this feature, we measure the standard deviation of the bin count within the histogram.

Our experimental results show that the above set of features provides an excellent texture description of the SRAM start-up data [66]. We found that memory chips with the same manufacturer and part number (specification) form separable clusters in feature space (the result is briefly presented in Sec. VI).

2) Analyzing Pattern Frequency: PUF outputs must not be biased toward a specific pattern. To capture the weakness of mPUF, we analyze the pattern (observed as the PUF outcome) frequency. Primarily, most of the applications use at least a 128-bit or 256-bit key for authentication or cryptographic operation. However, a 64-bit key may also be used in resource-constrained applications. A 128-bit key can form 2^{128} unique patterns (i.e., possible keys). Analyzing such a massive number of patterns is not always feasible due to the limitations of computational resources and time constraints. Furthermore, analyzing pattern distribution of 2^{128} CRPs requires a large source (memory chip) size ($>2^{128} \times 128$ bits $= 2^{92}$ Tbyte). which is not viable for commercial off-the-shelf (COTS) memory chips due to the limited address space. Therefore, we consider each PUF response as multi-word data and perceive PUF responses using the 16-bit word characteristics. We computed the relative frequency of the patterns formed by each of the 16-bit words of the memory and quantified the deviance of *Hamming weight* distribution as discussed in Sec. III-C. For an ideal SRAM-PUF, all patterns should have a uniform probability of occurrence (see Sec. III-C). However, from frequency analysis, we observe that some patterns have more probability of occurring than others. Note that in our experiment, we use 4-Mbit SRAM chips, where each memory chip provides 2^{18} 16-bit words.

3) Error Estimation of SRAM Signatures: According to our third assumption (see Sec. IV), the adversary must know the amount of error victim PUF can tolerate (i.e., the capability of the error correction scheme adopted for the victim system). However, the victim system does not reveal such information. To understand the error profile of the victim mPUF, we characterize errors of mPUFs that possess the same part number as the victim one. Error characterization is performed by collecting two set start-up data at two different operating conditions and computing *Hamming distance* between them. Once the error rate is estimated, the number of patterns (i.e., key search-space) can be reduced, as discussed in Sec. III-B (for algorithm, see Sec. V-B).

B. Attacking an SRAM PUF

In a traditional brute-force attack, an attacker guesses a possible set of keys and continuously tries them, hoping that one combination will match the original key. In such attacks, attackers do not have any prior knowledge about the key. However, some prior knowledge in the victim's device might provide some insights on pattern heuristics, enabling the attacker to order all possible patterns according to their probability of being matched. In this section, we mainly focus on characterizing frequent patterns and ordering them. From now on, we will frequently use two terms: *target device* and *train devices*. The *target device* is owned by the victim and targeted by the attacker. On the other hand, the *train devices* are possessed by the attacker (assumption 2 in Sec. IV), and they are similar (i.e., same manufacturer and part number) to the *target device*. Our proposed attacking scheme uses *train devices* to learn and characterize the *target device*.

We assume that the PUF response length is 64 ($n = 64$) in the rest of the paper for simplification. However, a 64-bit binary number can form 2^{64} possible unique patterns, which require, on average, 2^{63} attempts to discover the key using a simple brute-force attack [67]. In our proposed attacking scheme, we aim to reduce the number of attempts (i.e., reducing the search space) by analyzing *train devices* and ordering all possible 64-bit patterns by assigning a relative probability to each pattern.

Most of the SRAM chips use either 16-bit or 8-bit parallel interface. Hence, we need to concatenate 4 (for 16-bit memory) or 8 (for 8-bit memory) addresses to produce a 64-bit key. We use a 4-Mbit 16-bit SRAM memory chips for our experiment and assume that a 64-bit key is evaluated from 4 consecutive memory addresses. Therefore, we can generate a total of 65, 536 CRPs ($\ll 2^{64}$) from a 4-Mbit mPUF.

Ordering 2^{64} patterns from a 64-bit number is a difficult task. Therefore, we have fragmented the 64-bit key into four 16-bit words ($n' = 16$). In our proposed attacking scheme, we first estimate the error rate in each PUF response from the *train devices* accepted in the authentication/key re-generation phase. Then we observe the frequencies of all 16-bit patterns in *train devices*. In the next step, using the estimated error rate and relative frequency information of the 16-bit patterns, we have reduced the number of 16-bit patterns using a custom *covering code* algorithm and ordered them.

We use Algorithm 1 to estimate errors in PUF responses of the *target device*. To estimate the error, we first record all responses (corresponding to all possible challenges) of all *train devices* at two different valid operating conditions (e.g., at different voltage and/or temperature). Next, we compute errors using the *Hamming distance* between these two sets of responses. As the *train devices* are similar to the *target device*, the *train devices'* maximum error is the hard error limit of the *target device* (i.e., an attacker needs to produce a response within that error limit to make a successful attack). However, this assumption might seems too optimistic from the attacker side (i.e., the same maximum error limit of *train devices* and *target devices*). Hence, to relax the condition, an attacker may choose the ith percentile of the error distribution (distribution of $\mathcal{D}$ from Algorithm 1) and set the value as the estimated maximum error limit of the *target device* (i.e., the value of q).

Algorithm 1 Quantifying Error Limit During Key Re-generation/Authentication Phase

Data: Two sets of n-bit response data, $\mathcal{R}_{oc1}$ and $\mathcal{R}_{oc2}$ from *train devices* at two valid operating conditions, *op1* and *op2*.
$\mathcal{R}_{oc1} \leftarrow \{R^0_{oc1},\ R^1_{oc1},\ R^2_{oc1},\ \ldots\}$;
$\mathcal{R}_{oc2} \leftarrow \{R^0_{oc2},\ R^1_{oc2},\ R^2_{oc2},\ \ldots\}$;
Result: q, The number of errors accepted out of n-bit response.

```
1 begin
    /* Compute element-wise Hamming distance (d(·)) between R_oc1 and R_oc2 */
2   D ← {D_i | D_i = d(R^i_oc1, R^i_oc2)};
3   q ← max(D);
4 end
```

Next, we propose Algorithm 2 to order all n-bit patterns. In our proposed attacking scheme, an attacker uses pre-ordered n-bit patterns to perform a heuristic attack. In this heuristic attack, patterns from the top of the order have a higher probability of succeeding than the bottom of the order. Algorithm 2 generates the most significant unique t attempts of n bits. As we explained earlier (Sec. V-A.2), ordering an n-bit pattern is a difficult task; we first order all possible $2^{n'}$ patterns expressed by each n'-bit word. Note that the number of allowed erroneous bits in an n'-bit pattern is denoted by q', and the number of unique n'-bit patterns is denoted by t'. Later, these n'-bit patterns are joined to generate total t unique n-bit patterns.

First, all possible n' bit patterns are listed in a variable X (of length $2^{n'}$). Next, we compute a similarity matrix M of size $2^{n'} \times 2^{n'}$, where $M[i, j]$ is the *Hamming distance* between $X[i]$ and $X[j]$. We compute the histogram corresponding to each pattern in X by observing their frequency in all *train devices* and denoted with B_X (see line 9 of Algorithm 2). Next, the indices that would sort (in descending order) the B_X are computed and saved in idx (line 10). We, then, reorder X, B_X, and M (both row-wise and column-wise) according to idx and store them in X^s, B^s_X, and M^s, respectively. In line 15,

Algorithm 2 Reducing and Ordering n-Bit Patterns

Data: n, PUF response length.
n', Word length in SRAM memory.
q, Input from Algorithm 1.
t, Number of attempts (n-bit pattern).
Result: T, Serialized n-bit patterns; $len(T) \ll 2^n$.
// Initialization
1 $k \leftarrow \frac{n}{n'}$; // The number of words to represent n-bit response
2 $q' \leftarrow \frac{q}{k}$; // Allowed bit-error in n'-bit word.
3 $X \leftarrow \{0,\ 1,\ \ldots,\ 2^{n'}-1\}$; // Set of n'-bit words
4 $p_{n'} \leftarrow [\ ]$; // List of reduced n'-bit patterns
5 $w_p \leftarrow [\ ]$; // Weights of $p_{n'}$
6 $t' \leftarrow ceil(\sqrt[k]{t})$; // Required number of n'-bit patterns
/* Compute similarity matrix of n'-bit patterns */
7 $M \leftarrow \begin{pmatrix} d(0,0) & d(0,1) & \cdots & d(0,b_{2^{n'}-1}) \\ d(1,0) & d(1,1) & \cdots & d(1,b_{2^{n'}-1}) \\ \vdots & \vdots & \ddots & \vdots \\ d(b_{2^{n'}-1},0) & d(b_{2^{n'}-1},1) & \cdots & d(b_{2^{n'}-1},b_{2^{n'}-1}) \end{pmatrix}$;
8 **begin**
9 $B_X \leftarrow \{b_0,\ b_1,\ \ldots,\ b_{2^{n'}-1}\}$; // Histogram of X
10 $idx \leftarrow argsort(B_X)$; // Indices that sorts B_X
11 $X^s \leftarrow X[idx]$; // Sorted X
12 $B_X^s \leftarrow B_X[idx]$; // Sorted B_X
13 $M^s \leftarrow M[idx, :]$; // Sorted the row of M
14 $M^s \leftarrow M^s[:, idx]$; // Sorted the column of M
15 $M_b^s \leftarrow (M^s \le q')$; // $2^{n'} \times 2^{n'}$ Boolean matrix
16 $i \leftarrow 0$;
17 **while** $(len(X^s) > 0)$ && $(i < t')$ **do**
18 $p_{n'}.append(X^s[i])$;
19 $w \leftarrow M_b^s[i, :] \cdot B_X$; // Weight of $p_{n'}[i]$
20 $w_p.append(w)$;
/* Compute covered pattern indices by $X^s[i]$ */
21 $j \leftarrow args(M_b^s[i, :] == 1)$;
/* Remove covered elements from X^s & M_b^s */
22 $del\ X^s[j]$;
23 $del\ M_b^s[j, \ ]$;
24 $del\ M_b^s[:, \ j]$;
25 $i \leftarrow i + 1$;
26 **end**
/* Permute (with repetition) the indices of $p_{n'}$ vector by taking k elements */
27 $idx^p \leftarrow perm(\{0,\ 1,\ \ldots,\ len(p_{n'}) - 1\}, k)$;
28 $p_{n'}^k \leftarrow p_{n'}[idx^p]$; // Permute $p_{n'}$
29 $w_k \leftarrow w_p[idx^p]$; // Permute w_p
30 $w_k^p \leftarrow sum(w_k, row)$; // Each permutation's weight
31 $idx_{p,f} \leftarrow argsort(w_k^p)$; // Indices that sorts w_k^p
32 $T \leftarrow horstack(p_{n'}^k[idx_{p,f}, :])$;
33 **end**

we computed a Boolean matrix M_b^s, in which, $M_b^s[i, j] = 1$, if $M^s[i, j] \le q'$ and 0 otherwise. If $M_b^s[i, j] = 1$, then $X^s[j]$ is covered by $X^s[i]$ within q' bit error. From lines 17 to 26, we computed the reduced set of n'-bit pattern ($p_{n'}$) and corresponding weight (w). In ith iteration of the loop, we appended the most frequent n'-bit pattern ($X^s[i]$) in $p_{n'}$. We compute the weight of $p_{n'}[i]$ by accumulating only those bins in B_X^s that are codependent on the patterns covered by $p_{n'}[i]$ (or $X^s[i]$). In line 21, we compute all pattern indices that are covered by $X^s[i]$, and then we remove corresponding entries from X^s, and rows/columns from M_b^s. At line 27, we compute the special indices idx^p that generate all possible permutations of the input vector (in this case, indices of $p_{n'}$) by taking k elements (with repetition). A simple Cartesian product is used to do such kind of task. For example, if we consider a set,

$$\begin{aligned} A &= \{a_0, a_1, a_2\};\ \text{and corresponding indices,} \\ idx_A &= \{0, 1, 2\};\ \text{then,} \\ idx_A^p &= perm(idx_A, 2) = idx_A \times idx_A \\ &= \{(0,0), (0,1), (0,2), (1,1), (1,2), (2,2)\} \\ A[idx_A^p] &= \{(a_0, a_0), (a_0, a_1), (a_0, a_2), \\ &\quad (a_1, a_1), (a_1, a_2), (a_2, a_2)\} \end{aligned}$$

Now, using this idx^p, we compute the corresponding permutation of $p_{n'}^k$ and w_k. We, then, calculate w_k^p by summing the weights from each row of w_k. Next, we compute the indices $idx_{p,f}$ that sorts the w_k^p in descending order. Finally, we sort the row of $p_{n'}^k$ based on the indices $idx_{p,f}$, and horizontally stacked all binary patterns of the row in a single string to form an n-bit pattern (stored in T). The attacker can use this pre-ordered n-bit string in T to perform the attack.

Algorithm 2 can be explained using Fig. 5. We concatenate k n'-bit words from the memory chip to form an n-bit key ($n = k \times n'$). Here we assume that each of the n'-bit words are independent of each other. So, to attempt with t n-bit pattern, we only need the most frequent $t'(= \sqrt[k]{t})$ n'-bit words. Later, these n'-bit words are combined with different permutations to form t n-bit pattern.

Fig. 5. Constructing n-bit key from n'-bit word.

VI. Results and Analysis

To demonstrate the SRAM-based PUF vulnerability and potential attack from it, we collected SRAM start-up data from 139 4-Mbit commercial SRAM memory chips from four major manufacturers. We used these chips as the *train devices* (see Sec. V-B). We list all of these memory chips in Table I with the corresponding manufacturer and part number. These chips are also tagged with unique identifier, *Chip tag*, and will be used in the rest of the paper. To emulate the proposed attack, we collected data from another new chip from each group (not included in Table I) and used them as the *target devices* (see Sec. V-B).

We collected SRAM start-up data using an Arduino platform [68] at a nominal voltage (3.3V) and room temperature (25°C). The effect of noise was minimized by collecting the start-up data 19 times from each chip and combining them using the majority voting technique [69].

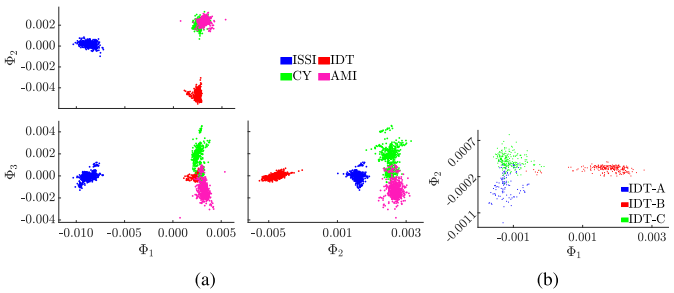


Fig. 6. Visualizing of SRAM samples in feature-space, by: (a) manufacturer and (b) part number. Dimension of feature-space is reduced using GDA [70].

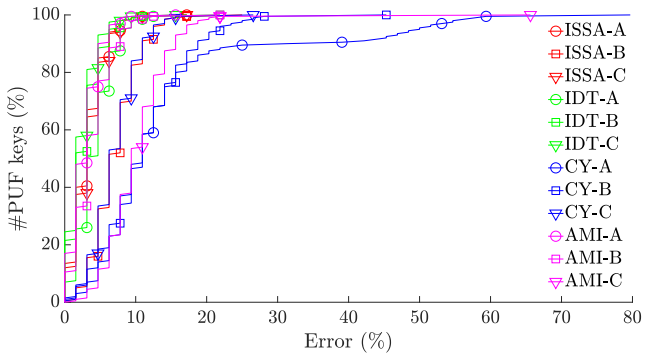


Fig. 7. SRAM chip error characterization.

TABLE I
LIST OF SRAM CHIPS USED AS *Train Devices*

Manufacturer[3]	Part number	Chip tag	#Chips
ISSI	IS61LV25616AL-10TL	ISSI-A	14
	IS61WV25616BLL-10TLI-TR	ISSI-B	6
	IS61LV25616AL-10TLI	ISSI-C	14
IDT	IDT71V416S10PHG8	IDT-A	7
	IDT71V416S12PHG8	IDT-B	14
	IDT71V416L15PHG8	IDT-C	14
CY	CY7C1041G18-15ZSXI	CY-A	14
	CY62147G30-55ZSXE	CY-B	10
	CY62146EV30LL-45ZSXIT	CY-C	8
AMI	AS7C34098A-10TCN	AMI-A	14
	AS7C34098A-10TIN	AMI-B	14
	AS6C4016-55ZIN	AMI-C	10

A. Correlation Among SRAM Signature Properties

To demonstrate the correlation among SRAM signatures, we explored the feature characteristics discussed in Sec. V-A.1. Since we have a small number of samples per group for classification/clustering, we segmented the whole memory into 16 256-Kbit chunks and treated them as individual chips. The memory features are extracted from each memory chunk. However, we reduced the feature dimension for better visualization using generalized discriminant analysis (GDA) [70]. The GDA is a supervised dimension reduction technique in which the high dimensional feature space is projected into a low dimensional feature space by preserving the maximum separability among the groups [70]. The lower feature dimension is nonlinearly related to the higher dimension (in this case, we used an *RBF* kernel function). The detailed discussion on GDA is out of the scope of this paper. Note that we used a random subset of *train devices* to learn the GDA model and used the rest of the *train devices* to construct Fig. 6.

Fig. 6a presents the SRAM signature separability in a 3-dimensional feature space (Φ_1, Φ_2, and Φ_3). The results show that memory chips from the same manufacturer cluster together and are separable at least in one 2-dimensional plot (Φ_1 vs. Φ_2, Φ_2 vs. Φ_3, or Φ_3 vs. Φ_1). In Fig. 6b, we present the SRAM signature separability based on their part number from a single manufacturer. The 2-dimensional feature space suggests that memory chips with the same part number form a cluster. This observation concludes that the SRAM memory signatures are highly dependent on the part number (i.e., specification). We also found a similar observation in our recent work on DRAM manufacturer identification [65]. Note that the non-deterministic random process variation causes some noisy points (see Fig. 6) in clusters. Additionally, we also have some overlapping regions in Fig. 6, which might be avoided by including more samples while learning the GDA model.

B. Error Characterization of SRAM Signature

According to our threat model (Sec. IV), the adversary needs to know the degree of errors accepted by the PUF protocol (Fig. 1) obtained from the *train devices*. The error is estimated using Algorithm 1, where the attacker needs to collect the PUF responses from the *train devices* at a minimum of two different operating conditions. In our experiment, we chose a random subset of the *train devices* at two operating conditions (at room temperature vs. at 45°C). Then we used those CRPs in Algorithm 1 to estimate the errors. Fig. 7 represents the error characteristic between these two sets of CRPs. The results show that SRAM chips from different manufacturers and part numbers have different levels of error tolerance. For example, the memory signatures from IDT-C are highly error-tolerant (see Fig. 7); almost 99.9% CRPs can be produced within ∼11% of errors. On the other hand, CY-A is highly susceptible to noise; only 50% of the CRPs can be reproduced within ∼11% error limit. To defend the system from various attacks (e.g., replay attack [71]), multiple CRPs are used over the device's lifetime. However, increasing the number of usable CRPs also requires increasing the number of acceptable erroneous bits. For example, for CY-A, CY-B, and AMI-C, using 99.9% of the CRPs requires to allow a large number of error bits. On the other hand, for IDT-C, 99.9% of the CRPs can be used by just allowing <11% errors. For better error estimation, the adversary can always collect start-up data at extreme operating conditions from the *train devices* and observe the noise performance. However, to keep our experiment simple, for Algorithm 2, we assumed 12.5% erroneous bits (1 out of 8 bit) is the

[3]ISSI: Integrated Silicon Solution, Inc.; IDT: Integrated Device Technology; CY: Cypress Semiconductor; AMI: Alliance Memory, Inc.
If reviewers ask, we will remove the chip information (manufacturers' name and part number) from the final version of the paper.

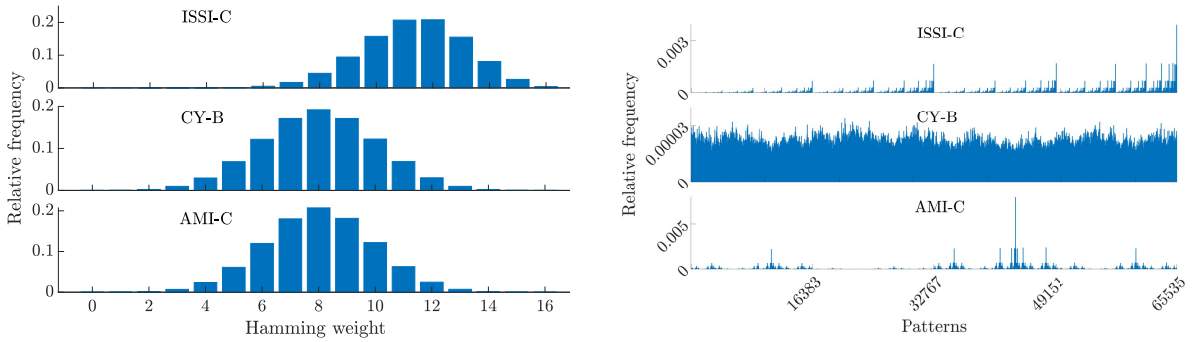


Fig. 8. Relative frequency of- (a) *Hamming weights*, and (b) 16-bit patterns.

TABLE II
16-BIT PATTERN CHARACTERISTICS AND VULNERABILITY TO HEURISTIC ATTACK

Chip tag	HW distribution (16-bit patterns)		β_C		#Reduced patterns (16-bit)	CRP_{rec} (%)
	μ	σ	μ	σ		
ISSI-A	10.49	1.90	0.66	0.02	1953	13.06
ISSI-B	7.92	2.03	0.50	0.04	1830	14.99
ISSI-C	11.30	1.83	0.71	0.02	1922	32.54
IDT-A	8.84	2.60	0.55	0.11	1747	8.70
IDT-B	8.96	2.10	0.56	0.03	1877	1.65
IDT-C	8.87	2.57	0.55	0.10	1799	13.77
CY-A	8.01	2.11	0.50	0.02	1694	0.11
CY-B	8.00	2.05	0.50	0.02	1741	0.23
CY-C	5.60	1.93	0.35	0.02	1923	11.51
AMI-A	7.99	2.08	0.50	0.03	1688	0.16
AMI-B	7.99	2.25	0.50	0.05	1740	0.13
AMI-C	8.01	1.92	0.50	0.02	1934	44.33

maximum acceptable limit by the PUF protocol, which is reasonable for most of the cases [62].

C. Attacking an SRAM PUF

Table II represents the silicon results of each memory group. In the second and third columns of Table II, the *Hamming weight* distribution of all 16-bit words from all *training devices* is presented. For the uniform possibility of all patterns, the mean and standard deviation of *Hamming weight* should be close to $\frac{16}{2} = 8$ and $\frac{\sqrt{16}}{2} = 2$ (see Sec. III-C and Appendix B). However, from Table II, we notice that not all SRAM signatures follow these criteria. Moreover, the distribution of *Hamming weight* does not solely guarantee the uniform probability of all patterns (see Appendix B), which is demonstrated in Fig. 8. Fig. 8a represents the distribution of the *Hamming weight* for different cases, and in Fig. 8b, we present the corresponding relative frequency of each 16-bit pattern (pattern 0 to 65535). For ISSI-C, the mean of the *Hamming weight* largely deviates from 8 (Fig. 8a), and the corresponding relative frequency plot (Fig. 8b) shows that a few 16-bit patterns have a higher probability of appearing than others. On the other hand, for CY-B and AMI-C, the mean and

Fig. 9. Spatial distribution of 0's and 1's- (a) CY-B, and (b) AMI-C.

standard deviation of the *Hamming weight* are close to 8 and 2 (Fig. 8a), respectively. This also indicates that the chips from CY-B and AMI-C are not biased toward "0" or "1". Such unbiased property is also clearly visible from Fig. 9, which presents the spatial distribution of 0's and 1's from CY-B and AMI-C using consecutive 1KByte memory space. Despite both CY-B and AMI-C exhibit almost an ideal *Hamming weight* distribution, some patterns from AMI-C have more probability of occurring than others (Fig. 8b). Hence, to ensure a uniform probability of each pattern, we recommend checking actual pattern distribution rather than focusing on *Hamming weight* distribution (see Appendix B).

Columns 4 and 5 of Table II represent the mean and standard deviation of β_C (Eqn. 2 and 3) computed from all possible challenges, C. For most memory groups, the mean value is ∼0.5, with a very small standard deviation. However, for ISSI-A, ISSI-C, and CY-C, the mean value of β_C largely deviates from 0.5, which can be explained by the *Hamming weight* distribution. For example, the mean *Hamming weight* of 16-bit patterns in ISSI-A is 10.49, i.e., the probability of getting "1" in such device is $\frac{10.49}{16} \approx 0.66$, which is exactly the mean value of β_C. This unique correlation of *Hamming weight* distribution and *bit-aliasing* is true for all SRAM groups presented in Table II. Such a correlation is expected as both the *Hamming weight* and *bit-aliasing* measure the probability of getting "1" (or "0") of a given bit location. If the distribution of "1" (or "0") is uniform across the whole memory chip and follows the same distribution over all chips, then the probability value calculated from the *Hamming weight* and the *bit-aliasing* should be the same.

Column 6 of Table II presents the reduced number of 16-bit patterns covering all 16-bit patterns within a 12.5% error limit. Note that the number of patterns is higher than the

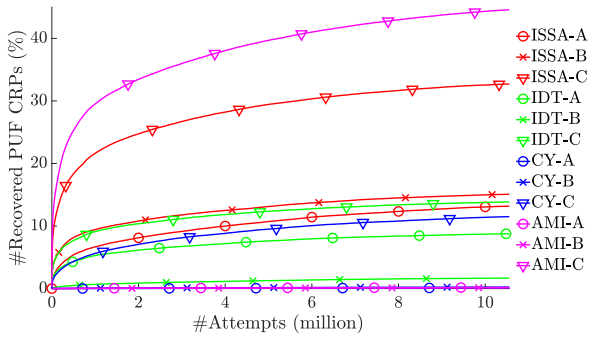


Fig. 10. Number of attempts vs. recovered CRPs.

range specified in [58] (<960). Because, in our Algorithm 2, we emphasize maximizing the probability of success for each attempt instead of ensuring the smallest covering set in the proposed heuristic attack. However, as only <2000 patterns cover all of the 16-bit patterns, the entropy of each 16-bit word reduces to $< \log_2(2000) < 11$. Hence, the allowance of 12.5% of errors (i.e., 2 out of 16-bit) reduces the entropy by >5 bits.

Column 7 of Table II presents the percentage of recovered CRPs using Algorithm 2 with 10 million (10×10^6) attempts. To generate these 10 million 64-bit unique patterns, we only use the most frequent 57 ($\approx \sqrt[k]{t} = \sqrt[4]{10 \times 10^6}$) 16-bit words. In each attempt, we use a different 64-bit pattern. For each group of memory, the number of attempts vs. the percentage of recovered CRPs is presented in Fig. 10. From Table II (column 7) and Fig. 10, we can divide our results into the following three categories.

1) *Category–1:* Chips from this category (IDT-B, CY-A, CY-B, AMI-A, AMI-B) are less vulnerable to our heuristic attack (success rate is <2%). Chips for this category possess almost a perfect distribution of Hamming weight (see columns 2 and 3 of Table II; $\mu \approx \frac{16}{2} = 8$ and $\sigma \approx \frac{\sqrt{16}}{2} = 2$) and maintain a nearly uniform distribution on their 16-bit pattern (e.g., CY-B in Fig. 8b).
2) *Category–2:* Using our proposed attacking algorithm, we revealed 8–15% CRPs from this category (ISSI-A, ISSI-B, IDT-A, IDT-C, CY-C). For chips from ISSI-A and CY-A, the consequence is directly understandable from the skewed *Hamming weight* distribution of the 16-bit pattern (μ deviated from the ideal value of 8; see columns 2 of Table II). However, although the chips from ISSI-B, IDT-A, IDT-C follows a nearly perfect Hamming weight distribution, 16-bit patterns of those chips deviate from the uniform distribution (not shown in Fig. 8b).
3) *Category–3:* Chips from this category are highly vulnerable to our proposed attack (ISSI-C and AMI-C). For those chips, we recovered 32.54% and 44.33% CRPs, respectively. ISSI-C is highly skewed from the *Hamming weight* distribution of the 16-bit pattern ($\mu = 11.30$, see columns 2 of Table II). Although, the vulnerability of AMI-C is not visible from the Hamming weight distribution ($\mu \approx 8$ and $\sigma \approx 2$), the actual pattern distribution (Fig. 8b) can capture it. For AMI-C, some 16-bits patterns are highly probable than others, making it easier to guess the possible value of the PUF outcome (see Fig. 8b).

Note that using our algorithm, we are able to recover, on average, ∼12% of the CRPs by using only 10 million attempts (i.e., we only used $<10^{-10}$% of all possible 64-bit patterns). As these 64-bit patterns are pre-computed and pre-ordered, the computational overhead of our proposed attack is negligible. However, in an ideal scenario (i.e., with equal probability of all 64-bit patterns and 0-bit error tolerance), finding a single CRP with a traditional brute-force attack is very low ($\approx 5.72 \times 10^{-11}$%, see Appendix C).

Our attacking algorithm can be further improved by extracting some spatial locality information from the *train devices*. In Sec. V-A.1 and Sec. VI-A, we presented that the SRAM start-up data characteristics are largely correlated with the features extracted from spatial locality information. Although it is quite challenging to uncover detailed spatial locality information from the start-up data due to the internal data and address scrambling [72], [73], we observe some periodic relationship between the start-up data and the logical address of the memory (especially, SRAM chips manufactured by the IDT). For example, Fig. 11a shows a particular case for IDT-C SRAM memory. Along the horizontal axis, we present the logical address of the 4-Mbit memory, and along the vertical axis, the percentage of logic "1" from the next consecutive 256 words. This plot shows that the number of 1's in this memory has a periodic nature (with a period of 16,384), which might come from several sources. Instead of complete randomization of address and data, the manufacturer may choose to quantize the whole memory in small segments and scramble those segments only can be one of the reasons. To improve our attacking scheme, we divide memory chips into two segments by drawing a magenta straight line (Fig. 11a). The logical addresses above the straight-line are grouped in segment 1, and under the straight-line are grouped in segment 2. Then, we applied the same Algorithm 2 to attack CRPs from those two segments independently (on the *target device*), and the corresponding result is shown in the 11b. Fig. 11b shows that a higher percentage of CRPs from segment 1 than segment 2 is recovered with the same number of attempts (17.59% of CRPs from segment 1 (red curve), 9.18% from segment 2 (green curve), and 13.39% on average (violet curve)). This result is expected, as the mean *Hamming weight* of patterns from segment 1 largely deviates from the ideal, i.e., 50%.

D. Attacking in the Presence of a Fuzzy Extractor

As we discussed in Sec. II-B, in many PUF applications, a fuzzy extractor is frequently used to correct PUF output and enhance security. To guarantee security, the fuzzy extractor must satisfy the condition provided by Eqn. 1. The fuzzy min-entropy from each group of SRAM chips is shown in Table III (calculated using a *training device*). Although fuzzy extractors constructed with fuzzy min-entropy can enhance the PUF security, the security of the fuzzy extractor still might not be up to the mark as expected. We mentioned in Sec. II-B that it is hard to quantify the probability associated with each input pattern of the fuzzy

TABLE III
FUZZY MIN-ENTROPY FOR EACH SRAM CHIP; GIVEN THAT, $length(R) = 16$ AND $q = 2$

Chip tag	ISSI-A	ISSI-B	ISSI-C	IDT-A	IDT-B	IDT-C	CY-A	CY-B	CY-C	AMI-A	AMI-B	AMI-C
$H_{\infty}^{fuzzy}(R)$	4.36	4.59	3.18	5.09	6.46	5.18	8.24	8.19	4.42	8.41	7.48	2.78

extractor from the experimental data. Such imperfection in statistical estimation arises from the uncertainty associated with the experiment. For example, for CY-B, a 16-bit input of a fuzzy extractor should produce ∼8-bit long outputs. To produce a 64-bit key, we require at least eight 16-bit words from the CY-B SRAM chip (i.e., total 128-bit input). Now, to achieve the same level of success as presented in Table II (0.23%), we require all possible combinations of the same $57(= t')$ 16-bit words (as explained in Sec. VI-C) to produce 128-bit input, where $k = \frac{128}{16} = 8$ (see Algorithm 2). So, even after using a properly designed fuzzy extractor for CY-B, a total of $t = (t')^k = 57^8 \approx 111$ trillion 128-bit raw input patterns would be sufficient to recover 0.23% of 64-bit CRPs. However, Eq. 7 (Appendix C) shows that 111 trillion trials would recover 0.0009% of 64-bit CRPs, while the PUF outcome is completely random. That is, even by following the recommended boundary condition of the fuzzy extractor; it might not be possible to achieve the target level of security.

VII. MITIGATION

We propose several countermeasures against the proposed heuristic attack in order to generate secure and robust signatures from SRAM chips.

A. Avoiding Error Correction Scheme

We have demonstrated that using the error correction scheme reduces entropy. Avoiding error correction schemes in PUF protocol narrows down the attack surface in two ways: i) attackers require to guess the CRPs with an exact match, ii) it prevents the attacker from reducing the number of patterns as we proposed in Algorithm 2. Researchers have proposed several robust SRAM cell design techniques (e.g., [74]) and selection techniques (e.g., [75]) to avoid error correction schemes. Unfortunately, the robust SRAM cell design methods usually require a modified mask layout and might require satisfying additional design rules; whereas, the robust SRAM cell selection technique might not be suitable in the presence of memory address/data scrambling [72], [73]. Moreover, they may still require error correction code; however, applying such methods requires correcting a smaller number of erroneous bits, significantly reducing the attack surface.

B. Using Properly Designed Fuzzy Extractor

Although fuzzy extractors are popularly used for correcting errors and improving security, many researchers proposed simpler versions of fuzzy extractors that use a fixed-length input to produce the fixed-length output [34]–[37], regardless of the fuzzy min-entropy (Eq. 1). However, based on our findings, we recommend using a fuzzy extractor that strictly follows the fuzzy min-entropy boundary condition. Additionally, we also recommend using conservative probabilistic estimation of each input pattern of the fuzzy extractor. For example, one can use the upper bound of 99% confidence interval of estimated probability to avoid the uncertainty associated with the estimation. However, such pessimistic estimation might reduce the number of CRP drastically; in the worst case, it might not be possible to generate any CRP from a given type of chip. Moreover, fuzzy-extractors are vulnerable to side-channel attack [76].

C. Pattern Distribution-Aware CRPs

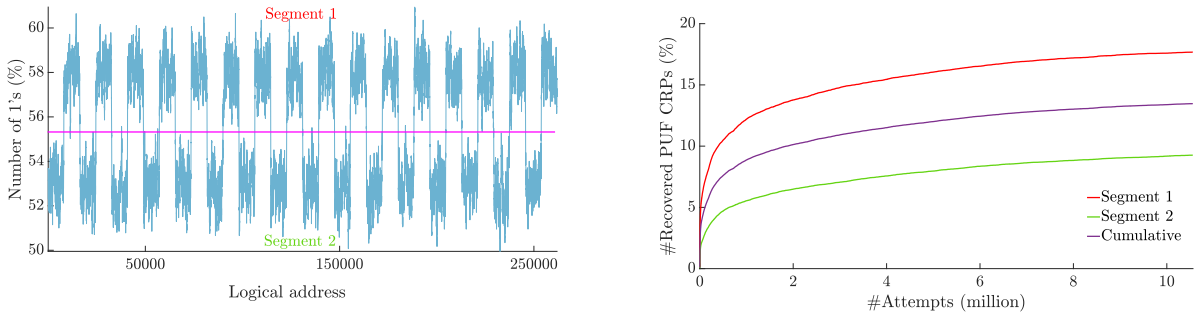
In Sec. III-C and Sec. VI-C, we emphasize uniform distribution of PUF outcome (response) for all possible challenges (address) to maximize the entropy of the PUF output. The uniform distribution of patterns prevents the attacker from ordering patterns, as we proposed in Algorithm 2. For example, in Fig. 8, the 16-bit pattern distribution for CY-B is more uniform than ISSI-C and AMI-C. As a result, Table II shows that CY-B is significantly less vulnerable to our proposed attack than the ISSI-C or AMI-C. Hence, we recommend choosing a subset of SRAM addresses (challenges) at the enrollment/registration phase to make a uniform distribution of all possible responses (i.e., addresses with highly repeated patterns should be discarded).

D. Locality-Aware CRPs

In Sec. VI-C, we noticed that PUF responses generated from some specific logical address segments might be more vulnerable to other segments (Fig. 11). So, choosing a less vulnerable logical address segment to generate CRPs might help avoid the proposed heuristic attack. However, if the attacker can reveal the SRAM layout (Sec. VIII), this task might become more challenging. Such knowledge of chip layout can enable the attacker to redesign Algorithm 2 more efficiently. Hence, for critical applications, the SRAM vendor is responsible for providing a guideline on PUF usage as they have the precise knowledge of SRAM physical layout.

VIII. FUTURE WORK

Our proposed attacking scheme can be further sophisticated by revealing the chip layout of the *target device*. In Sec. III-A, we assume that the *training device* and *target device* may produce correlated signatures as they share unique architectural, layout, and process variations. An attacker can learn those variations from his *training devices* and exploit the information to attack the victim's device. Invasive or semi-invasive methods can be used to reverse engineer the chip layout [77]–[80]. Reverse engineering of chips allows the attacker to extract exact architectural and layout design and enable the attacker to compute spatial distribution resulting from the architectural layout. Furthermore, the deterministic components of the process variations can also be reconstructed to some extent [54], [55]; however, learning the exact spatial distribution of process variation for a memory chip is difficult due to internal address randomization [72] and data scrambling [73]. Nonetheless, using these spatial distributions, the attacker can form multiple groups of CRPs based on spatial location and generate independent sets of ordered patterns



(a) (b)

Fig. 11. Improved heuristic attack on IDT-C type SRAM by observing spatial locality of patterns- (a) start-up data dependency on logical address, (b) number of attempts vs. recovered CRPs from different segments of the memory.

(as described in Algorithm 2) for each group. We expect that such techniques will improve our proposed attacking scheme, and we will investigate that experimentally in our future work.

IX. Conclusion

In this paper, we used the vulnerabilities of SRAM PUF and the weakness of existing metrics to mount a non-invasive attack. Traditionally, the mPUF response is believed to be a direct consequence of random process variation. However, we demonstrated that other deterministic factors such as architectural, layout, and the systematic process variations could significantly impact mPUF outputs and produce a common trait among all memory chips. Furthermore, we also demonstrated that the error correction scheme might severely weaken the PUF security, and the traditional *Hamming weight*-based metric of PUF "goodness" does not ensure the PUF quality. If we do not consider these factors carefully, the unclonable PUF will not be secure anymore.

To avoid such an attack, we emphasize reducing error correction code usage, checking the pattern uniformity on memory-based signatures, and locality aware challenge-response pairs (CRPs). Although our proposed countermeasures might reduce the total number of CRPs, we believe that such techniques will significantly improve the PUF security.

Appendix A
Hamming Weight Distribution

An n-bit binary string can form a total of 2^n outcomes. For proper randomness, each of the outcomes should have an equal probability of $\frac{1}{2^n}$. Alternatively, the probability of getting a specific symbol ("0" or "1") at t^{th} bit location of the string is $\frac{1}{2}$ (assuming the outcome is not biased to any symbol).

Next, consider a random n-bit string (S_i) of *Hamming weight* k (i.e., out of n bits, k bits of the string is "1"). So, the probability of getting such string as the PUF outcome,

$$p(HW = k) = \sum_i p(S_{i,HW=k}) = \binom{n}{k} \times (\frac{1}{2^n})$$

$$\Rightarrow p(HW = k) = \binom{n}{k} \times (\frac{1}{2})^k \times (\frac{1}{2})^{n-k} \quad (4)$$

Eqn. 4 is the probability mass function (PMF) of the Binomial distribution, where the second and third term in the right-hand side comes from the fact that the probability of getting "0" or "1" in any specific position of the bit string is $\frac{1}{2}$.

TABLE IV
Impact of Standard Deviation on Gaussian Approximation

HW	$\#S_{HW}$	$\sigma = \sqrt{2}$			$\sigma = 2\sqrt{2}$			$\sigma = \frac{\sqrt{2}}{2}$		
		P	f	f_r	P	f	f_r	P	f	f_r
0	1	0.01	1.32	1.32	0.05	13.28	13.28	0.00	0.00	0.00
1	8	0.03	7.61	0.95	0.08	20.57	2.57	0.00	0.02	0.00
2	28	0.10	26.57	0.95	0.11	28.12	1.00	0.01	2.65	0.09
3	56	0.22	56.24	1.00	0.13	33.92	0.61	0.21	53.13	0.95
4	70	0.28	72.22	1.03	0.14	36.11	0.52	0.56	144.43	2.06
5	56	0.22	56.24	1.00	0.13	33.92	0.61	0.21	53.13	0.95
6	28	0.10	26.57	0.95	0.11	28.12	1.00	0.01	2.65	0.09
7	8	0.03	7.61	0.95	0.08	20.57	2.57	0.00	0.02	0.00
8	1	0.01	1.32	1.32	0.05	13.28	13.28	0.00	0.00	0.00

Appendix B
Impact of Hamming Weight Distribution

While approximating a Binomial distribution of *Hamming weight* with a Gaussian curve, the probability of getting "0" or "1" bit in each bit location must be $p = \frac{1}{2}$. This condition can be further simplified with the following two conditions:

1) The mean *Hamming weight*, $\mu = np = \frac{n}{2}$; where n is the key length.
2) The standard deviation of *Hamming weight*, $\sigma = \sqrt{np(1-p)} = \frac{\sqrt{n}}{2}$.

In many previous works on PUF, the second condition is overlooked. If this condition is not properly satisfied, an attacker can obtain the key more easily with a heuristic attack.

Consider an example PUF with $n = 8$. The *Hamming weight* must be within the range $[0, n]$, and the total number of possible patterns is $N = 2^n = 256$. The mean and standard deviation of the approximated Gaussian distribution, $\frac{n}{2} = 4$ and $\frac{\sqrt{n}}{2} = \sqrt{2}$. With this setup, a simple demonstration of standard deviation is presented in Table IV. In this table, the column "HW" represents all possible *Hamming weights* achieved by an 8-bit pattern. The column "$\#S_{HW}$" presents the number of possible 8-bit patterns that correspond to the *Hamming weight* presented in the column "HW". For example, 70 different 8-bit patterns are possible with a *Hamming weight*

of 4. The column "P" under $\sigma = \sqrt{2}$ presents the calculated probability density value from the Gaussian approximation with $\mu = 4$ and $\sigma = \sqrt{2}$. In the next column (column "f"), the relative occurrence[4] of 8-bit patterns with corresponding *Hamming weight* is estimated ($P \times N$). In the next column, the ratio of corresponding "f" and "#(S_{HW})" is calculated. In the next three columns, we again calculate "P", "f", and "f_r" with $\sigma = 2\sqrt{2}$. In the last three columns, "P", "f", and "f_r" parameters are recalculated with $\sigma = \frac{\sqrt{2}}{2}$. For an accurate estimation, the value of "f_r" column should be close to 1. From the table, it is evident that, if $\sigma = \frac{\sqrt{n}}{2}$ ($\sigma = \sqrt{2}$), then Gaussian distribution can provide the approximate distribution of the *Hamming weight*. However, if the Gaussian distribution is fitted with a $\sigma > \frac{\sqrt{n}}{2}$ (in the table, $\sigma = 2\sqrt{2}$), then the patterns with extreme *Hamming weight* (i.e., close to the 0 or 8) will have a higher probability to occur ($f_r > 1$ for those patterns). On the other hand, if the Gaussian distribution is fitted with a $\sigma < \frac{\sqrt{n}}{2}$ (in the table, $\sigma = \frac{\sqrt{2}}{2}$), then the patterns with mean *Hamming weight* (i.e., close to the 4) will have a higher probability to occur ($f_r > 1$ for those patterns). Note that satisfying the condition of μ and σ might not be sufficient and will not guarantee that all possible patterns have an equal probability to be the PUF outcome. For example, in an 8-bit PUF, if the pattern "01010101" appears for 70 different challenges and all other patterns with $HW = 4$ do not appear at any challenge, then the condition on μ and σ still might get fulfilled. However, such PUF still might be susceptible to a heuristic attack. In summary, the impacts of the *Hamming weight* distribution are listed below:

- For a given unbiased n-bit PUF, all possible n-bit patterns should have a uniform probability to be the PUF outcome. Therefore, the corresponding *Hamming weight* distribution of PUF responses (for all possible challenges) should follow a Binomial distribution (with $\mu = \frac{n}{2}$, and $\sigma = \frac{\sqrt{n}}{2}$). This distribution can be approximated with a Gaussian distribution with the same mean and standard deviation.
- If the *Hamming weight* does not obey the above distribution, the PUF will be more susceptible to a heuristic attack.
- The Gaussian distribution (with $\mu = \frac{n}{2}$, and $\sigma = \frac{\sqrt{n}}{2}$) in *Hamming weight* does not guarantee the uniform probability of all patterns. Hence, we also recommend checking uniformity on pattern distribution.

APPENDIX C
PUF KEY RECOVERING USING BRUTE-FORCE ATTACK

If a random source (i.e., SRAM chip) can produce m n-bit keys (denoted as S_m), the probability of m keys being unique is defined as:

$$p_{unique} = \prod_{j=0}^{m-1} \frac{2^n - j}{2^n} \tag{5}$$

[4]Relative occurrence means that if the total number of CRPs produced by the target memory chip is exactly $N = 2^n$ (i.e., the total area under the distribution curve is N), how many times may one expect a pattern with the specific *Hamming weight*. This value should be close to the value presented in the "#(S_{HW})" column.

Hence, the probability of having a duplicate key, $p_{duplicate} = 1 - p_{unique}$. In this paper, we have used $n = 64$ and $m = 65,536$, which gives $p_{duplicate} \approx 1.164 \times 10^{-10}$. For simplicity, we can eliminate such cases of having duplicate patterns in S_m.

Now, let's assume we have randomly selected t unique n-bit patterns (denoted as S_t), where $t \ll 2^n$. Hence, the probability of matching l pattern from S_t with S_m:

$$p(S_t \cap S_m = l) = \binom{m}{l} \times \left[\prod_{j=1}^{l} \frac{t - (j-1)}{2^n - (j-1)} \right] \tag{6}$$

Therefore, the expected number of recovered keys from t patterns can be expressed with the following equation:

$$E(\#key) = \sum_{j=1}^{m} [j \times p(S_t \cap S_m = j)] \tag{7}$$

With $l = 10$ million, $E(\#key) \approx 5.72 \times 10^{-11}\%$.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their valuable feedback on their article.

REFERENCES

[1] S. Sutar, A. Raha, and V. Raghunathan, "D-PUF: An intrinsically reconfigurable DRAM PUF for device authentication in embedded systems," in *Proc. Int. Conf. Compil., Architectures Synth. Embedded Syst. (CASES)*, 2016, pp. 1–10, Art. no. 12. [Online]. Available: https://dl.acm.org/doi/10.1145/2968455.2968519

[2] D. E. Holcomb, W. P. Burleson, and K. Fu, "Initial SRAM state as a fingerprint and source of true random numbers for RFID tags," in *Proc. Conf. RFID Secur.*, 2007. [Online]. Available: http://www.rfid-cusp.org/publication.html

[3] S. Kiamehr *et al.*, "The impact of process variation and stochastic aging in nanoscale VLSI," in *Proc. IEEE Int. Rel. Phys. Symp. (IRPS)*, Apr. 2016, pp. 1–6.

[4] J. Guajardo *et al.*, "FPGA intrinsic PUFs and their use for IP protection," in *Proc. Cryptograph. Hardw. Embedded Syst.*, 2007, pp. 63–80.

[5] B. M. S. B. Talukder, B. Ray, D. Forte, and M. T. Rahman, "PreLatPUF: Exploiting DRAM latency variations for generating robust device signatures," *IEEE Access*, vol. 7, pp. 81106–81120, 2019.

[6] Microsemi. (Mar. 2016). *Using SRAM PUF System Service in SmartFusion2*. Accessed: Feb. 19, 2021. [Online]. Available: http://www.is.gd/microsemiPUF

[7] NXP Semiconductors. (Feb. 2019). *LPC55Sxx Usage of the PUF and Hash Crypt to AES Coding*. Accessed: Jun. 6, 2020. [Online]. Available: http://www.is.gd/nxpPUF

[8] T. Lu, R. Kenny, and S. Atsatt. *Secure Device Manager for Intel Stratix 10 Devices Provides FPGA and SOC Security*. Accessed: Jun. 6, 2020. [Online]. Available: http://www.is.gd/IntelSecureDeviceMngr

[9] I. ID. *Protecting a Device's Root Secrets With SRAM PUF*. Accessed: Jun. 6, 2020. [Online]. Available: http://www.is.gd/intrinsicidPUF

[10] Xilinx. (Aug. 2018). *Developing Tamper-Resistant Designs With ZYNQ Ultrascale+ Devices*. Accessed: Jun. 6, 2020. [Online]. Available: http://www.is.gd/xilinxPUF

[11] G.-J. Schrijen and C. Garlati, "Physical unclonable functions to the rescue," in *Proc. Embedded World*, 2018. [Online]. Available: http://www.intrinsic-id.com/wp-content/uploads/2018/05/Physical-Unclonable-Functions-to-the-Rescue-A-way-to-establish-trust-in-silicon-Schrijen-Garlati-Embedded-World-2018.pdf

[12] C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas, "Physical unclonable functions and applications: A tutorial," *Proc. IEEE*, vol. 102, no. 8, pp. 1126–1141, Aug. 2014.

[13] U. Ruhrmair *et al.*, "PUF modeling attacks on simulated and silicon data," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 11, pp. 1876–1891, Nov. 2013.

[14] T. McGrath, I. E. Bagci, Z. M. Wang, U. Roedig, and R. J. Young, "A PUF taxonomy," *Appl. Phys. Rev.*, vol. 6, no. 1, Mar. 2019, Art. no. 011303.

[15] U. Rührmair and D. E. Holcomb, "PUFs at a glance," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2014, pp. 1–6.

[16] M. V. Dijk and U. Rührmair, "Protocol attacks on advanced PUF protocols and countermeasures," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2014, pp. 1–6.

[17] U. Ruhrmair and M. van Dijk, "PUFs in security protocols: Attack models and security evaluations," in *Proc. IEEE Symp. Secur. Privacy*, May 2013, pp. 286–300.

[18] U. Rührmair, C. Jaeger, and M. Algasinger, "An attack on PUF-based session key exchange and a hardware-based countermeasure: Erasable PUFs," in *Proc. Financial Cryptogr. Data Secur.*, 2012, pp. 190–204.

[19] D. Karakoyunlu and B. Sunar, "Differential template attacks on PUF enabled cryptographic devices," in *Proc. IEEE Int. Workshop Inf. Forensics Secur.*, Dec. 2010, pp. 1–6.

[20] D. Merli *et al.*, "Side-channel analysis of PUFs and fuzzy extractors," in *Proc. Trust Trustworthy Comput.*, 2011, pp. 33–47.

[21] Q. Ma, C. Gu, N. Hanley, C. Wang, W. Liu, and M. O'Neill, "A machine learning attack resistant multi-PUF design on FPGA," in *Proc. 23rd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2018, pp. 97–104.

[22] N. A. Anagnostopoulos, T. Arul, M. Rosenstihl, A. Schaller, S. Gabmeyer, and S. Katzenbeisser, "Low-temperature data remanence attacks against intrinsic SRAM PUFs," in *Proc. 21st Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2018, pp. 581–585.

[23] S. Zeitouni, Y. Oren, C. Wachsmann, P. Koeberl, and A.-R. Sadeghi, "Remanence decay side-channel: The PUF case," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 6, pp. 1106–1116, Jun. 2016.

[24] D. Nedospasov, J.-P. Seifert, C. Helfmeier, and C. Boit, "Invasive PUF analysis," in *Proc. Workshop Fault Diagnosis Tolerance Cryptogr.*, Aug. 2013, pp. 30–38.

[25] C. Helfmeier, C. Boit, D. Nedospasov, and J.-P. Seifert, "Cloning physically unclonable functions," in *Proc. IEEE Int. Symp. Hardw.-Oriented Secur. Trust (HOST)*, Jun. 2013, pp. 1–6.

[26] A. Roelke and M. R. Stan, "Attacking an SRAM-based PUF through wearout," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2016, pp. 206–211.

[27] W. Che, F. Saqib, and J. Plusquellic, "PUF-based authentication," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2015, pp. 337–344.

[28] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proc. 44th ACM/IEEE Design Autom. Conf.*, Jun. 2007, pp. 9–14.

[29] C.-H. Chang, Y. Zheng, and L. Zhang, "A retrospective and a look forward: Fifteen years of physical unclonable function advancement," *IEEE Circuits Syst. Mag.*, vol. 17, no. 3, pp. 32–62, 3rd Quart., 2017.

[30] B. Fuller, L. Reyzin, and A. Smith, "When are fuzzy extractors possible?" *IEEE Trans. Inf. Theory*, vol. 66, no. 8, pp. 5282–5298, Aug. 2020.

[31] B. Fuller, L. Reyzin, and A. Smith, "When are fuzzy extractors possible?" in *Advances in Cryptology—ASIACRYPT*. Berlin, Germany: Springer, 2016, pp. 277–306.

[32] R. Canetti *et al.*, "Reusable fuzzy extractors for low-entropy distributions," in *Advances in Cryptology—EUROCRYPT*. Berlin, Germany: Springer, 2016, pp. 117–146.

[33] X. Boyen, "Reusable cryptographic fuzzy extractors," in *Proc. 11th ACM Conf. Comput. Commun. Secur. (CCS)*, 2004, pp. 82–91.

[34] H. Kang, Y. Hori, T. Katashita, M. Hagiwara, and K. Iwamura, "Cryptographie key generation from PUF data using efficient fuzzy extractors," in *Proc. 16th Int. Conf. Adv. Commun. Technol.*, Feb. 2014, pp. 23–26.

[35] M. Hiller, L. Kürzinger, and G. Sigl, "Review of error correction for PUFs and evaluation on state-of-the-art FPGAs," *J. Cryptograph. Eng.*, vol. 10, no. 3, pp. 229–247, Sep. 2020.

[36] M.-D. Yu and S. Devadas, "Secure and robust error correction for physical unclonable functions," *IEEE Des. Test Comput.*, vol. 27, no. 1, pp. 48–65, Jan./Feb. 2010.

[37] M.-D. Yu *et al.*, "Lightweight and secure PUF key storage using limits of machine learning," in *Cryptographic Hardware and Embedded Systems—CHES* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2011, pp. 358–373.

[38] Y. Yilmaz, S. R. Gunn, and B. Halak, "Lightweight PUF-based authentication protocol for IoT devices," in *Proc. IEEE 3rd Int. Verification Secur. Workshop (IVSW)*, Jul. 2018, pp. 38–43.

[39] A. Maiti, V. Gunreddy, and P. Schaumont, *A Systematic Method to Evaluate and Compare the Performance of Physical Unclonable Functions*. New York, NY, USA: Springer, 2013, pp. 245–267.

[40] M. T. Rahman, A. Hosey, Z. Guo, J. Carroll, D. Forte, and M. Tehranipoor, "Systematic correlation and cell neighborhood analysis of SRAM PUF for robust and unique key generation," *J. Hardw. Syst. Secur.*, vol. 1, no. 2, pp. 137–155, 2017.

[41] A. Babaei and G. Schiele, "Physical unclonable functions in the Internet of Things: State of the art and open challenges," *Sensors*, vol. 19, no. 14, p. 3208, Jul. 2019.

[42] C. Premalatha, K. Sarika, and P. M. Kannan, "A comparative analysis of 6T, 7T, 8T and 9T SRAM cells in 90 nm technology," in *Proc. IEEE Int. Conf. Electr., Comput. Commun. Technol. (ICECCT)*, Mar. 2015, pp. 1–5.

[43] R. Rollini, J. Sampson, and P. Sivakumar, "Comparison on 6T, 5T and 4T SRAM cell using 22 nm technology," in *Proc. IEEE Int. Conf. Electr., Instrum. Commun. Eng. (ICEICE)*, Apr. 2017, pp. 1–4.

[44] T. Bauer and J. Hamlet, "Physical unclonable functions: A primer," *IEEE Security Privacy*, vol. 12, no. 6, pp. 97–101, Nov. 2014.

[45] F. Moradi, S. K. Gupta, G. Panagopoulos, D. T. Wisland, H. Mahmoodi, and K. Roy, "Asymmetrically doped FinFETs for low-power robust SRAMs," *IEEE Trans. Electron Devices*, vol. 58, no. 12, pp. 4241–4249, Dec. 2011.

[46] A. Kawasumi *et al.*, "A single-power-supply 0.7V 1 GHz 45 nm SRAM with an asymmetrical unit-ß-ratio memory cell," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 2008, pp. 382–622.

[47] G. Apostolidis, D. Balobas, and N. Konofaos, "Design and simulation of 6T SRAM cell architectures in 32 nm technology," *J. Eng. Sci. Technol. Rev.*, vol. 9, no. 5, pp. 145–149, 2016.

[48] F. Liu, "A general framework for spatial correlation modeling in VLSI design," in *Proc. 44th ACM/IEEE Design Autom. Conf.*, Jun. 2007, pp. 817–822.

[49] J. Zhang and S. Gupta, "SRAM array yield estimation under spatially-correlated process variation," in *Proc. IEEE 23rd Asian Test Symp.*, Nov. 2014, pp. 149–155.

[50] K. Huang, N. Kupp, C. Xanthopoulos, J. M. Carulli, and Y. Makris, "Low-cost analog/RF IC testing through combined intra- and inter-die correlation models," *IEEE Design Test*, vol. 32, no. 1, pp. 53–60, Feb. 2015.

[51] K. Huang, N. Kupp, J. M. Carulli, and Y. Makris, "Process monitoring through wafer-level spatial variation decomposition," in *Proc. IEEE Int. Test Conf. (ITC)*, Sep. 2013, pp. 1–10.

[52] L. Cheng, P. Gupta, C. J. Spanos, K. Qian, and L. He, "Physically justifiable die-level modeling of spatial variation in view of systematic across wafer variability," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 3, pp. 388–401, Mar. 2011.

[53] N. Boynton, A. Pomerene, A. Starbuck, A. Lentine, and C. T. DeRose, "Characterization of systematic process variation in a silicon photonic platform," in *Proc. IEEE Opt. Interconnects Conf. (OI)*, Jun. 2017, pp. 11–12.

[54] K. Meng and R. Joseph, "Process variation aware cache leakage management," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, 2006, pp. 262–267.

[55] A. Agrawal, A. Ansari, and J. Torrellas, "Mosaic: Exploiting the spatial locality of process variation to reduce refresh energy in on-chip eDRAM modules," in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2014, pp. 84–95.

[56] A. Garg and T. T. Kim, "Design of SRAM PUF with improved uniformity and reliability utilizing device aging effect," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Jun. 2014, pp. 1941–1944.

[57] P. R. J. Ostergard, "Upper bounds for q-ary covering codes," *IEEE Trans. Inf. Theory*, vol. 37, no. 3, pp. 660–664, May 1991.

[58] Z. Zhang, "Linear inequalities for covering codes. I. Pair covering inequalities," *IEEE Trans. Inf. Theory*, vol. 37, no. 3, pp. 573–582, May 1991.

[59] W. Burr *et al.*, "Electronic authentication guideline," Nat. Inst. Standards Technol. (NIST), Gaithersburg, MD, USA, Tech. Rep. sp 800-63-2, Jun. 2017.

[60] H. Kalouti, D. E. Lazic, and T. Beth, "On the relation between distance distributions of binary block codes and the binomial distribution," *Annales Des Télécommunications*, vol. 50, nos. 9–10, pp. 762–778, Sep. 1995.

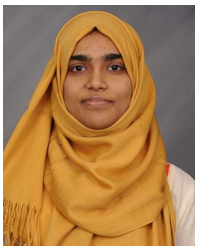
[61] M. L. Boas, *Mathematical Methods in the Physical Sciences*, 3rd ed. Hoboken, NJ, USA: Wiley, Jul. 2005, p. 762.

[62] G.-J. Schrijen and V. van der Leest, "Comparative analysis of SRAM memories used as PUF primitives," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2012, pp. 1319–1324.

[63] J. Woo and H. K. Kim, "Survey and research direction on online game security," in *Proc. Workshop SIGGRAPH Asia (WASA)*, 2012, pp. 19–25.

[64] P. Eckersley, "How unique is your web browser?" in *Proc. Privacy Enhancing Technol.*, 2010, pp. 1–18.
[65] B. M. S. B. Talukder, V. Menon, B. Ray, T. Neal, and M. T. Rahman, "Towards the avoidance of counterfeit memory: Identifying the DRAM origin," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, Dec. 2020, pp. 111–121.
[66] P. Wu, B. S. Manjunath, S. Newsam, and H. D. Shin, "A texture descriptor for browsing and similarity retrieval," *Signal Process., Image Commun.*, vol. 16, nos. 1–2, pp. 33–43, Sep. 2000.
[67] H. Marco-Gisbert and I. Ripoll, "Preventing brute force attacks against stack Canary protection on networking servers," in *Proc. IEEE 12th Int. Symp. Netw. Comput. Appl.*, Aug. 2013, pp. 243–250.
[68] *Arduino Due*. Accessed: Jul. 3, 2020. [Online]. Available: http://www.is.gd/arduinoDue
[69] M. Bhargava, C. Cakir, and K. Mai, "Reliability enhancement of bi-stable PUFs in 65 nm bulk CMOS," in *Proc. IEEE Int. Symp. Hardw.-Oriented Secur. Trust*, Jun. 2012, pp. 25–30.
[70] G. Baudat and F. Anouar, "Generalized discriminant analysis using a kernel approach," *Neural Comput.*, vol. 12, no. 10, pp. 2385–2404, 2000.
[71] Y. Mo and B. Sinopoli, "Secure control against replay attacks," in *Proc. 47th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Sep. 2009, pp. 911–918.
[72] A. J. van de Goor and I. Schanstra, "Address and data scrambling: Causes and impact on memory tests," in *Proc. 1st IEEE Int. Workshop Electron. Design, Test Appl.*, Jan. 2002, pp. 128–136.
[73] P. Ehlig and S. Pezzino. (Nov. 2017). *Error Detection in SRAM*. Accessed: Jul. 3, 2020. [Online]. Available: http://www.is.gd/SRAMErrDetect
[74] J.-W. Jang and S. Ghosh, "Design and analysis of novel SRAM PUFs with embedded latch for robustness," in *Proc. 16th Int. Symp. Qual. Electron. Design*, Mar. 2015, pp. 298–302.
[75] K. Xiao, M. T. Rahman, D. Forte, Y. Huang, M. Su, and M. Tehranipoor, "Bit selection algorithm suitable for high-volume production of SRAM-PUF," in *Proc. IEEE Int. Symp. Hardware-Oriented Secur. Trust (HOST)*, May 2014, pp. 101–106.
[76] D. Merli, D. Schuster, F. Stumpf, and G. Sigl, "Side-channel analysis of PUFs and fuzzy extractors," in *Trust and Trustworthy Computing*. Berlin, Germany: Springer, 2011, pp. 33–47.
[77] S. Blythe, B. Fraboni, S. Lall, H. Ahmed, and U. de Riu, "Layout reconstruction of complex silicon chips," *IEEE J. Solid-State Circuits*, vol. 28, no. 2, pp. 138–145, Feb. 1993.
[78] G. Masalskis and R. Navickas, "Reverse engineering of CMOS integrated circuits," *Elektronika ir Elektrotechnika*, vol. 88, no. 8, pp. 25–28, 2008.
[79] S. E. Quadir *et al.*, "A survey on chip to system reverse engineering," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 1, pp. 1–34, Dec. 2016.
[80] J. Kumagai, "Chip detectives [reverse engineering]," *IEEE Spectr.*, vol. 37, no. 11, pp. 43–48, Nov. 2000.

B. M. S. Bahar Talukder (Graduate Student Member, IEEE) received the bachelor's degree from Bangladesh University of Engineering and Technology, Dhaka, Bangladesh. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Florida International University. His primary research interests include hardware security and reliability, secured computer architecture, machine learning application in system security, and emerging memory technologies.



Farah Ferdaus (Graduate Student Member, IEEE) received the B.Sc. degree in electrical and electronic engineering from Bangladesh University of Engineering and Technology in 2015 and the M.S. degree in electrical and computer engineering (ECE) from the University of New Hampshire in 2018. She is currently pursuing the Ph.D. degree in ECE with Florida International University. Her research interests include performance enhancement and security solutions of emerging memories, privacy and security issues of existing memories, and wireless communications and networks.

Md Tauhidur Rahman (Senior Member, IEEE) received the Ph.D. degree in computer engineering from the University of Florida in 2017. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Florida International University (FIU). Before joining FIU, he was an Assistant Professor with The University of Alabama in Huntsville (UAH). His current research interests include hardware security and trust, memory systems, machine learning applications, embedded security, and reliability.