# TMVS: Threshold-based Majority Voting Scheme for Robust SRAM PUFs

Sara Faour, Mališa Vučinić, Filip Maksimovic, David C Burnett, Paul Mühlethaler, Thomas Watteyne, Kristofer Pister

# TMVS: Threshold-based Majority Voting Scheme for Robust SRAM PUFs

Sara Faour*, Mališa Vučinić*, Filip Maksimovic*, David C. Burnett†,
Paul Muhlethaler*, Thomas Watteyne*, Kristofer Pister‡

*Inria, Paris
†Portland State University
‡University of California, Berkeley
e-mail: first.last@inria.fr*, dburnett@pdx.edu†, ksjp@berkeley.edu‡

***Abstract*—Using a Physically Unclonable Function (PUF) for extracting a secret key from the unique submicron structure of an integrated circuit, rather than storing it in non-volatile memory, provides notable advantages such as physical unclonability and tamper resistance. SRAM PUFs offer the advantage of reusing memories that already exist in many resource-constrained embedded devices, by leveraging the unique start-up values of memory cells. However, certain SRAM cells may exhibit bit-flipping due to noise, temperature and voltage fluctuations. In this paper, we propose a straightforward yet effective method to mitigate PUF error rates. This method uses a majority voting decoder to rectify errors only in selected SRAM patterns, determined by a predefined threshold. Our scheme is simpler than other error correction codes (ECCs) based solutions. Even for the simplest ECCs, our method can provide better leftover entropy, e.g., more than 3x improvement, at the cost of worse error correction capability and memory overhead. TMVS can reach much smaller bit-error rate values compared to existing solutions based on majority voting, e.g., 360x improvement. TMVS has also minimal pre-processing runtime and cost with respect to existing pre-processing techniques like preselection and hardening.**

***Index Terms*—SRAM PUF, bit flipping, secret key, reliability.**

## I. Introduction

All keyed cryptographic primitives rely on the secrecy of the key used to encrypt or sign a given message. Conventional secure key storage schemes use non-volatile memory (NVM) technologies to store cryptographic secrets. This method introduces additional manufacturing process steps, costs, footprint while decreasing security because of possible readout and cloning attacks. Physical Unclonable Functions (PUFs) have appeared as a promising solution for secret key storage, boosting superior performance, lower cost, and tamper resistance.

Due to submicron process variations inherent in manufacturing, every transistor within an integrated circuit has slightly distinct physical properties. These properties can be measured, and since the process variations are uncontrollable, they result in features that cannot be replicated, even by the manufacturer. Therefore, a PUF serves as a silicon biometric for an electronic device, which can generate a unique, repeatable and unpredictable response to an applied stimulus or challenge. These properties have made PUFs very useful in secure key generation and storage. The Arbiter PUF [1] and Ring Oscillator PUF [2] were proposed based on delay measurements. The Butterfly PUF [3], D Flip-Flop PUF [4], Buskeeper PUF [5] and SRAM PUF [6] were proposed relying on start-up values of memory cells.

SRAM PUF is the only type of PUFs that can be completely implemented in software without requiring a dedicated circuit. This PUF leverages the unpredictable power-up value of SRAM cells to provide the PUF response, with the corresponding address serving as the challenge. This unpredictability arises from inherent random process variations in the relative strengths of the two cross-coupled inverters within each cell, caused by random-dopant fluctuations, line-edge roughness and other factors [7]. Consequently, upon power-up, some SRAM cells initialize to logical '0' while others to logical '1'. This unique pattern of '0's and '1's across each SRAM can then be used for secure key generation.

The reliability of the SRAM cells is one of the challenges of SRAM PUF. The response of SRAM PUF is noisy because some SRAM cells are unstable, which means that their values are not always the same during multiple power-up cycles. The power-up state in these cells is highly sensitive to random noise, temperature and voltage variations. To generate secret keys, the response for each SRAM cell should be highly reliable under different operating conditions, with a bit-error rate (BER) close to zero.

We classify the existing methods used to solve the problem of robustness on SRAM PUFs into software-based and non software-based approaches. Non software-based approaches either select stable SRAM cells during testing, e.g. multiple measurements and power resets of the SRAM at different voltage/temperature conditions [8], or strengthen the response of SRAM cells by applying reverse burn-in aging [8]. However, software-based approaches are mainly based on error correction codes (ECCs), e.g. BCH code, whether as a base solution [9], or as a complementary technique [10]. ECCs correct the erroneous bits and reduce the bit-error rate to a specific level. Based on these two main classes, combinations

of different approaches have been also exploited [11].

*Motivation.* We aim at developing an *efficient* solution to extract a reliable secret key from an SRAM PUF. By "efficient", we refer to hardware resources, cost and runtime of the implementation, in large networks of resource-constrained devices. We need our solution implementation to be as compact as possible, without the need to perform pre-processing steps as in non software-based approaches. All of these steps can add cost and time overhead that is unacceptable for many applications, including large networks of wireless sensor nodes. Additionally, we want the new approach to be as low-complexity as possible compared to ECC-based solutions. This is a key requirement that makes our work fundamentally different from other SRAM PUF implementations.

*Contribution.* A naive majority voting decoder produces bit 1 if the sequence of bits has more '1' bits, and bit 0 if it has more '0' bits. Although this decoder has low-complexity, it fails to decrease the PUF error rate, since a single noisy cell can change the majority decision when a pattern has close number of zeros and ones. In our approach, we exclude these SRAM patterns causing error decoding from PUF, and choose only the patterns with small or large number of ones according to a threshold. The selected patterns, even when noisy, undergo the majority voting decoder and produce highly reliable bits. This new method neither requires knowing the positions of the most stable SRAM cells, nor using complex ECC techniques. The contribution of this paper is three-fold:

1) We develop a new comprehensive methodology, called TMVS, to increase the SRAM PUF reliability by selecting certain SRAM patterns to be used as PUF.
2) We theoretically analyze the approach by deriving the probability of decoding error and the probability of finding a pattern that respects the determined threshold.
3) We discuss the proposed approach with respect to existing solutions in the literature.

*Paper Organization.* The remainder of this paper is organized as follows. Section II surveys related work. Section III provides a background on SRAM PUF and fuzzy extractors. Section IV describes the TMVS approach, showing its parameters, requirements and phases. Section V analyzes the approach theoretically. Section VI discusses TMVS with respect to existing techniques. Section VII concludes this paper and presents avenues for future work.

## II. Related Work

Several mechanisms have been proposed in the literature to reduce cell instability and increase reliability of SRAM PUFs.

*Error Correction Codes (ECC).* ECCs such as the repetition code, BCH, Reed-Muller, or Golay codes [9] are used to construct fuzzy extractors and reduce the BER to a specific level, so that the SRAM array can be directly used as a PUF. A powerful ECC provides strong error correcting capability but also leads to an excessively large hardware overhead, which is inadequate in some hardware resource-limited applications.

*Preselection.* Highly stable cells are selected during testing and used as PUF cells, to ensure high reliability. A stable cell has high probability of having the same power-up value. The overall hardware overhead is negligible since only the address of the selected cell requires to be stored in NVM. However, testing cells requires multiple measurements and power resets at different voltage/temperature conditions [8], [12], during the enrollment of a key, which is an unacceptable overhead for many applications. Even when some methods operate under a single voltage/temperature condition, these methods require either precise control of the power-off duration of the power supply [13], posing implementation challenge for advanced CMOS technology; or regulation of the voltage ramp rate [14], [15], an option dependent on chip design. Moreover, some complex circuits such as adjustable regulators could be needed.

*Soft Decision Maximum-Likelihood (SDML).* Similarly to preselection methods, multiple reliability measures are required in SDML to calculate individual bits error probabilities. SDML decoder uses these values to select the code word that was most likely transmitted [16]. It achieves the best possible error-correcting performance, but generally at a decoding complexity exponential in the number of key bits per block, and a large quantity of helper data stored in NVM. Even when SDML uses a single measure [10], ECCs will be still required.

*Spatial Majority Voting (SMV).* The idea of the SMV scheme [17] is about performing majority voting only on the subgroup of bits that constitutes the majority of total bits; e.g. for the group of bits '11100', the majority voting is applied only on the first three bits. However, this method is not sufficient by itself, and other techniques such as preselction or ECC are still needed to reach nearly 100% reliability.

*Hardening.* Reverse burn-in aging is applied to the SRAM cells to strengthen their response and harden their direction [8]. The cells aged with 0 (storing 0) will have a bias of 1, and the cells aged with 1 will lean towards 0 bias. This method somewhat improves the PUF stability, but is not sufficient on its own, and ECC is still required afterwards. Additionally, it is an expensive procedure in terms of tester runtime and cost. Recently, a new burn-in technique that can reach nearly 100% reliable SRAM PUF cells was explored [18]. However, it requires modifying the design of the SRAM cells, and this increases the design cost and time, and also limits occasional reuse of functional SRAMs as PUFs.

## III. Preliminaries

In this section, we review SRAM PUF and introduce some metrics and statistical functions that will serve as base for our approach construction. Later, we review the fuzzy

extractor, a technique based on error correction codes (ECCs) for improving SRAM PUF reliability using software only.

### A. SRAM PUF

When an SRAM is initially powered up, each cell acquires a '0' or a '1' logic value. A well-known circuit design for an SRAM cell is the 6-transistor SRAM cell. In an ideal SRAM cell, if all transistors are identical, the cell is perfectly balanced. In the absence of an asymmetric electrical noise, such a cell has a random 50% probability of acquiring either a '0' or a '1' state at power-up. However, in nanometer-scale technologies, because of uncontrollable small random manufacturing variations, no two transistors in an SRAM cell are truly identical in practice. Therefore, when the SRAM memory has been powered up, the process variation along with the noise and environment variation will classify the cells into unstable and stable cells. Stable cells will have their preferred and consistent state, either a '0' or a '1'. Conversely, the final state of unstable cells is determined by the noise present in the circuit and by temperature and voltage variation.

Due to the noisy nature of SRAM PUF responses, two readings of the SRAM cells at start-ups will produce two different but closely related responses $R$ and $R'$. The measure of closeness can be defined via the Hamming distance $d_H(R, R')$; the number of bits that differ between two bit strings. The raw PUF bits have a bit-error rate $p_e$, and many techniques in the literature focused on increasing SRAM PUF reliability and decreasing $p_e$. The target bit-error rate is $10^{-6}$, to make PUF responses adequate for key generation.

$$\text{bino}(k, n, p) = \binom{n}{k} p^k (1-p)^{n-k} \tag{1}$$

The bits in an SRAM PUF are derived from physically distinct circuit components, thus, assuming that the PUF bits are independent and identically distributed (*i.i.d*) is reasonable [19]. However, some SRAM PUFs present dependency between physically neighboring memory cells [20]. Hence, bias and correlation between SRAM cells should be tested in advance for specific circuit and technology [21]. We denote by $p$ the probability for a cell to have a '1' value, hence, perfectly unbiased SRAM memory cells will have $p = \frac{1}{2}$. Assuming PUF bits are *i.i.d*, the Hamming weight of a bit string, defined as the number of non-zero bits, follows a binomial distribution, with the probability mass function given in (1), where $n$ is the number of bits in the bit string, and $k$ the number of occurring '1'. Furthermore, the Hamming distances between different raw bit strings result in also a binomial distribution [22]. The binomial distribution has a mean $\mu = np$ and standard deviation $\sigma = \sqrt{np(1-p)}$.

### B. Fuzzy Extractor

PUF responses can not be used directly as a key in a cryptographic primitive for two reasons: they are noisy and not uniformly distributed. To deal with the first issue, at least one of the techniques presented in Sec. II has to be used to generate reliable keys. Mitigating the second issue requires applying a privacy amplification or randomness extraction primitive, e.g. hash functions, and this is out of scope of our study.

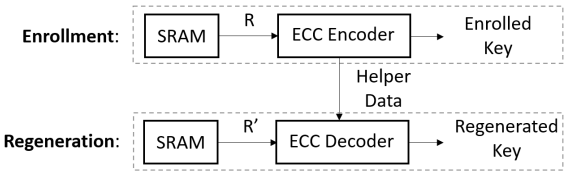


Fig. 1: ECC-based key generator of SRAM PUF.

Fuzzy extractor is a conventional approach based on ECCs and used to enhance the reliability of SRAM PUF and reach a target bit-error rate less than $10^{-6}$. Each SRAM PUF response $R$ is a sequence of $n$ bits. An ECC code $\mathcal{C}$ with parameters $[n, k, t]$ is used to correct the noisy response $R'$, where $n$ is also the length of the code, $k$ is the dimension of the input data, i.e. key bits, and $t$ is the number of errors it can correct. The error correcting capability $t$ is derived from the smallest Hamming distance $d_{\min}$ between any two different code words in $\mathcal{C}$ as $t = \lfloor \frac{d_{\min}-1}{2} \rfloor$.

There are two phases in a fuzzy extractor, as shown in Fig. 1: enrollment and regeneration. During the enrollment, the ECC code performs the encoding by using a large amount of raw data from the SRAM array. The output of encoding is the secret key $K$ and helper data $W$, where a code word $C_S$ is chosen at random from $\mathcal{C}$ and $W = C_S \oplus R$. This approach is known as "code-offset construction". The helper data is public and can be stored in any NVM. During the regeneration phase, the user will exploit the helper data and the new regenerated (noisy) PUF data $R'$ to recover the secret key.

$$P_{error} = \sum_{i=t+1}^{n} \binom{n}{i} p_e^i (1-p_e)^{n-i} \tag{2}$$

The probability that a string of $n$ bits has more than $t$ bit errors is given in (2), where $p_e$ is the raw bit-error rate, and once the number of errors exceeds $t$, the ECC can not correct them, thus, the SRAM PUF fails to extract the correct key. Notice that the desired value of $P_{error}$ determines the required error correcting capability $t$, and thus, the size of the code.

Different ECCs have been used to build robust SRAM PUFs. Using repetition code alone requires large PUF responses, using Golay code suffers high error probability, and BCH code has high decoding complexity. To mitigate these issues, Bösch*et al.* [9] proposed to use a concatenated code to reduce the size of the PUF response and the complexity for hardware implementation. These concatenated codes have as inner code a conventional error correcting code such as BCH, Reed-Muller, or Golay code, and as outer code a repetition code. Besides the hardware overhead, fuzzy extractors have

another drawback: the helper data bits can leak information about the extracted secret key [23]. Thus, helper data should be carefully generated.

## IV. Threshold-based Majority Voting Scheme

In this section, we show the core idea behind our approach based on the majority voting. We explain how to choose the threshold, which is crucial in our method. We present our approach using a simple use-case, then we describe the approach in a more general way. The goal of this approach, called TMVS, is to extract from an SRAM PUF a reliable key with error probability $10^{-6}$, a value assumed to be adequate for any realistic application [9].

### A. Majority Voting

The naive method to perform majority voting on a sequence of bits, is to extract bit 1 if the sequence has more '1' bits, and bit 0 if it has more '0' bits. Only sequences with odd number of bits are used in such approach to prevent ties. Unfortunately, this approach increases the PUF error rate as shown in [17] instead of reducing the error rate. This refers to the fact that a single noisy cell can change the majority decision for a full pattern. When the number of '0' and '1' bits in an SRAM pattern is very close, this approach fails immediately. Take for example the pattern "01100", the majority of bits is 0, however, if the first bit flips, the majority will become 1, thus, a decoding error occurs. In fact, when an SRAM memory is unbiased $p = \frac{1}{2}$, the Hamming weight of an SRAM pattern follows a binomial distribution with a mean equal to half of its length $n$ as explained in Sec. III-A. Therefore, many SRAM patterns have close number of zeros and ones, leading to a bad performance for the naive method.

When we exclude these SRAM patterns with Hamming weight close to $\frac{n}{2}$ from the PUF, the performance of the majority voting decoder increases significantly. Consequently, to decrease the error decoding probability, we should set a safety threshold on the Hamming weight of SRAM patterns selected to be used as PUF. SRAM patterns with close number of zeros and ones will be excluded from PUF according to a certain interval bounded by a lower and higher thresholds. The way we choose these thresholds is detailed next.

### B. Threshold Choice

The number of errors $n_e$ occurring in a pattern of $n$ *i.i.d* PUF bits with probability $p_e$, follows a binomial distribution bino$(n_e, n, p_e)$ defined in (1). The binomial can be approximated by the normal distribution if both $np_e > 5$ and $n(1 - p_e) > 5$. Assuming $p_e = 0.15$, $n$ should be larger than 33 to satisfy these conditions. This approximation allows us to apply the empirical rule, stating that around 68% of the error values are within one standard deviation ($\sigma_e$) of the mean ($\mu_e$), 95% of the values are within two standard deviations ($2\sigma_e$) and 99.7% are within three standard deviations ($3\sigma_e$).

In the context of our study, this statistical rule aims at predicting the number of possible erroneous bits occurring within any PUF response for $n$ cells with respect to a previous response. More precisely, we can deduce that the number of flipping bits between any two different SRAM PUF responses of the same cells, falls within the interval $[\mu_e - 3\sigma_e, \mu_e + 3\sigma_e]$ with a probability of 0.997. Thus, the maximum number of possible flipping bits is $\lceil \mu_e + 3\sigma_e \rceil$ bits with a probability of 0.997, for a response of length $n$. This maximum bound will be denoted as $N_{e,\max}(3\sigma_e)$. Therefore, by taking a higher coefficient of $\sigma_e$, we can find the interval within which around 100% of the error values fall, thus, $N_{e,\max}$ will be an accurate estimate for the maximum number of possible bit flips.

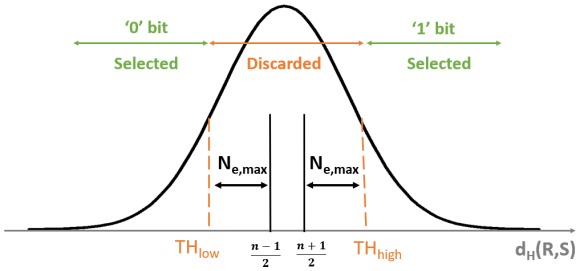


Fig. 2: Illustration for the selection region determined by thresholds on the Hamming distance $d_H(R, S)$ distribution.

In our approach, we select SRAM patterns in a way that even if the maximum number of bit flips $N_{e,\max}$ occurs, we remain able to extract exactly the same bits using majority voting. Hence, an SRAM pattern with Hamming weight far from $np$ by a margin of $N_{e,\max}$, can be decoded successfully. Consequently, only sequences with Hamming weight smaller than $np - N_{e,\max}$ or larger than $np + N_{e,\max}$ should be selected, these are respectively the lower and higher thresholds used in our approach.

*Requirements.* The diversity in chip designs leads to having different SRAM characteristics. Thus, in practical implementation of TMVS, an estimation of the worst-case BER $p_e$ and the average mean of bits $p$ of the used SRAMs is required.

### C. Basic Approach

*Codebook.* We start with a simple use-case of our approach where we choose a single code word $S_0$ equal to a sequence of $n$ zeros. Thus, the codebook $\mathcal{C}$ contains a single element. The Hamming distance between any SRAM pattern and $S_0$ is equivalent, in this case, to the Hamming weight of that pattern.

*Enrollment.* An SRAM response $R$ with $n$ raw bits is selected to be used as PUF if the Hamming distance $d_H(R, S_0)$ satisfies one of the following conditions:

$$d_H(R, S_0) \leq \lfloor np \rfloor - N_{e,\max} \quad (3)$$

$$d_H(R, S_0) \geq \lceil np \rceil + N_{e,\max} \quad (4)$$

We define the lower threshold in (3) as $TH_{\text{low}}$, and the higher threshold in (4) as $TH_{\text{high}}$, and we write (3) and 4 as follows:

$$d_H(R, S_0) \leq TH_{low} \quad (5)$$

$$d_H(R, S_0) \geq TH_{\text{high}} \quad (6)$$

Satisfying the first condition (5) implies that the response $R$ has a majority of '0' bits, and a '0' key bit can be extracted based on majority voting. For the second condition (6), a '1' key bit is extracted. These conditions are illustrated in Fig. 2 with respect to $d_H(R, S_0)$ that follows a binomial distribution as mentioned in IV-A, for $p = \frac{1}{2}$. To extract a full key of length $N_K$, we should be able to find $N_K$ SRAM patterns of length $n$ each, satisfying (5) or (6). If the SRAM PUF memory starts at address $x$, then the first tested pattern will be the pattern starting at $x$ and ending at $x + n - 1$. If the pattern was selected, then the next tested pattern starts at $x + n$, otherwise, if the pattern was discarded, we test the next pattern starting at $x+1$. Afterwards, the starting address of each *eligible* (selected) SRAM pattern is stored as helper data in any NVM.

*Regeneration.* To reconstruct the full key of length $N_K$, the helper data is used to identify the addresses of the selected patterns. A new SRAM PUF response $R'$ of a selected pattern is noisy with respect to its value obtained during enrollment. For each selected pattern, we calculate the Hamming distance $d_H(R', S_0)$, then we apply the majority voting decoder according to the following rule:

$$d_H(R', S_0) \leq \lfloor np \rfloor \rightarrow \text{bit key = '0'} \quad (7)$$

$$d_H(R', S_0) \geq \lceil np \rceil \rightarrow \text{bit key = '1'} \quad (8)$$

The first case (7) implies that the majority of bits in a pattern remains '0', so a '0' key bit is extracted. Otherwise, a key bit of '1' is generated. We can notice that even if the maximum number of bit flips $N_{e,\max}$ occurs, the worst case $d_H(R', S_0)$ will be equal to $\frac{n-1}{2}$ or $\frac{n+1}{2}$, with $p = \frac{1}{2}$. These values can be correctly decoded as well.

### D. General Approach

The previously described use-case uses a single code word $S_0$. This implies that during enrollment, a large number of SRAM patterns will be discarded before finding a pattern satisfying (5) or (6). Thus, the probability of selection for a pattern is very small, and a large SRAM size is required. A numerical example will be provided in Sec. V.

To increase the probability of selection for a pattern, more code words should be considered, so that the patterns excluded for a certain code word, could be selected with respect to another code word. Therefore, when a codebook $\mathcal{C}$ has multiple code words, the same approach described previously applies with a small difference during enrollment: for each SRAM pattern, the Hamming distance is calculated with respect to different code words, and the first code word satisfying (5) or (6) is assigned to it, as shown in Fig. 3. Consequently, the helper data includes both the address of the selected pattern and the index of its corresponding code word.

In fact, the larger the codebook, the larger the helper data size, because each code word index requires at least $\log_2(\text{nb. of code words})$ bits to be stored. Thus, a trade-off exists between the size of required SRAM PUF - quantified using the probability of selection - and the size of generated helper data. Since we need to generate reliable key bits with error probability of $10^{-6}$, depending on the raw PUF error rate $p_e$, it becomes an optimization problem to choose the best codebook and thresholds in terms of hardware resources, number of SRAM bits required, performance, etc.

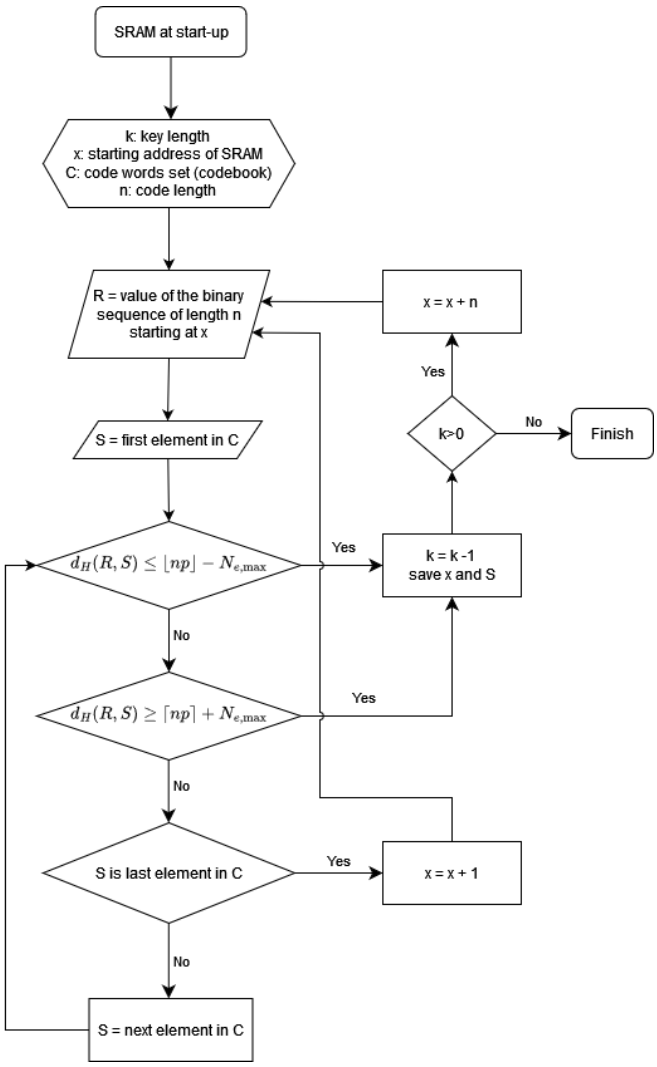


Fig. 3: Flowchart of general TMVS.

## V. Theoretical Analysis

In this section, we derive the theoretical formula for the probability of selecting a pattern according to fixed thresholds. Also, we derive the probability of decoding error, leading to regenerate a wrong key bit, assuming PUF cells are *i.i.d*; an assumption considered reasonable as discussed in Sec. III-A.

### A. Selection Probability

We define $P_{\text{select}}$ as the probability for any SRAM pattern $R$ to be selected with respect to any code word $S$. It is in fact the probability for $d_H(R, S)$ to satisfy (5) or (6), meaning

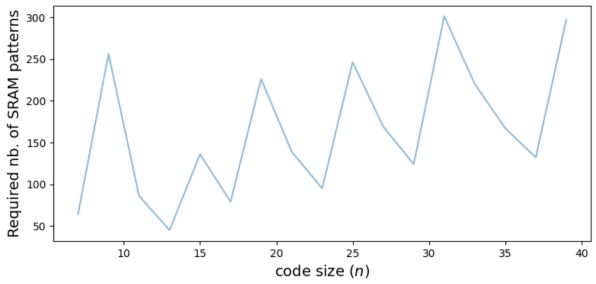


Fig. 4: The maximum number of discarded SRAM patterns before finding a single pattern satisfying (5) or (6) for different code size $n$, with $p_e = 0.08$, $p = 0.5$ and $N_{e,\max}(3\sigma_e)$.

that a pattern has at most $TH_{low}$ or at least $TH_{high}$ matching bits with $S$. This probability is given by:

$$P_{\text{select}} = \sum_{i=0}^{TH_{low}} \binom{n}{i} p^i (1-p)^{n-i} + \sum_{i=TH_{high}}^{n} \binom{n}{i} p^i (1-p)^{n-i} \quad (9)$$

This probability refers to a single code word $S$, hence, deriving the total probability of selection with respect to the full codebook $\mathcal{C}$ with $Card(\mathcal{C})$ code words requires knowing its construction. The total probability will be equal to $P_{\text{select}} \cdot Card(\mathcal{C})$ only if the code words do not overlap, meaning that no SRAM pattern can be assigned to more than one code word. Thus, the Hamming distance between every two different code words should be at least $d_{\min} = 2 \cdot TH_{low} + 1 = n - 2N_{e,\max}$.

*Numerical Example.* For $n = 35$, $p_e = 0.08$ and $p = 0.5$, we have $N_{e,\max}(3\sigma_e) = 8$, thus only SRAM patterns with $d_H(R, S_0) \leq 9$ or $d_H(R, S_0) \geq 26$ are selected, and $P_{\text{select}} = 0.006$. This implies that to deterministically find a single good pattern and select it, at most $\frac{1}{P_{\text{select}}} \approx 167$ different patterns might be discarded before. $P_{\text{select}}$ is very small, thus, a very large number of SRAM patterns will be discarded before finding an *eligible* pattern as shown in Fig. 4. However, considering more code words, can increase significantly the selection probability for any pattern.

### B. Error Decoding Probability

We define $P_{\text{error}}$ as the probability for a selected pattern to be wrongly decoded using the majority voting decoder. In the following, we denote by capital letters such as $N_e$ and $D_{\text{enr}}$ the random variables, and by small letters their values. The probability of $n_e$ errors to occur in a response of $n$ bits is:

$$P_{N_e}(n_e) = \binom{n}{n_e} p_e^{n_e} (1-p_e)^{n-n_e} \quad (10)$$

When a pattern $R$ is selected, the probability for it to have $d_H(R, S) = d_{\text{enr}}$ during enrollment, is derived as follows:

$$P_{D_{\text{enr}}}(d_{\text{enr}}) = \frac{1}{P_{\text{select}}} \cdot \binom{n}{d_{\text{enr}}} p^{d_{\text{enr}}} (1-p)^{n-d_{\text{enr}}} \quad (11)$$

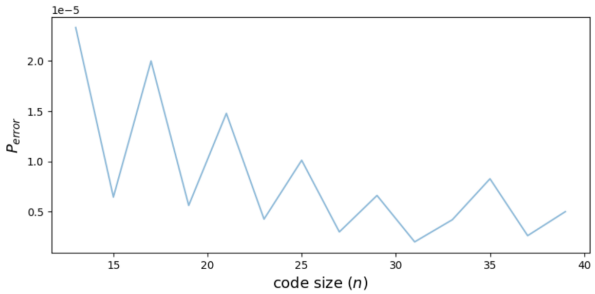


Fig. 5: Error decoding probability for different code size $n$, with $p_e = 0.08$, $p = 0.5$ and $N_{e,\max}(5\sigma_e)$.

Given its Hamming distance $d_{\text{enr}}$ during enrollment, a noisy pattern $R'$ will be wrongly decoded if the number of bit flips occurring is strictly larger than $t = \lfloor |np - d_{\text{enr}}| \rfloor$, the error correcting capability of TMVS. Therefore the probability for a wrong decoding to occur knowing $d_{\text{enr}}$ is:

$$P_{N_e,\, D_{\text{enr}}}(N_e \geq t+1 | d_{\text{enr}}) = \sum_{n_e=t+1}^{n} P_{N_e}(n_e) \quad (12)$$

Finally, the total probability of error decoding for any obtained Hamming distance $d_{\text{enr}}$ at enrollment is calculated as follows:

$$P_{\text{error}} = \sum_{d_{\text{enr}}=0}^{TH_{low}} P_{D_{\text{enr}}}(d_{\text{enr}}) P_{N_e,\, D_{\text{enr}}}(N_e \geq t+1 | d_{\text{enr}}) + \sum_{d_{\text{enr}}=TH_{high}}^{n} P_{D_{\text{enr}}}(d_{\text{enr}}) P_{N_e,\, D_{\text{enr}}}(N_e \geq t+1 | d_{\text{enr}}) \quad (13)$$

*Numerical Example.* For $n = 35$, $p_e = 0.15$ and $p = 0.5$, we will have $N_{e,\max}(5\sigma_e) = 16$, thus only SRAM patterns with $d_H(R, S_0) \leq 1$ or $d_H(R, S_0) \geq 34$ are selected, and $P_{\text{error}} = 2.83 \cdot 10^{-6}$. For a smaller $p_e$ value, we show the plot of $P_{\text{error}}$ as a function of code size $n$ in Fig. 5.

## VI. Discussion

Based on the approach description in Section IV, and the formulas in Section V, we compare the performance of TMVS to existing techniques presented in Sec. II. We discuss few criteria including the decoding complexity, memory overhead, entropy loss, bit-error rate (BER), enrollment time and cost.

*Non Software-Based Approaches.* Approaches such as preselection and hardening would either require multiple measures under all possible conditions, or controlling the power supply, or applying reverse burn-in techniques. These requirements lead to a longer time and higher cost required for enrolling each PUF. Therefore, this could be very challenging when enrolling millions of devices in production lines. Our approach requires an estimation of the worst-case BER and the average mean of bits (bias). These required parameters are not specific for every single cell as required for other approaches. In our case, an estimation over several chips is sufficient, thus no expensive characterization effort

is required. The estimation can be done using stochastic concentration theory [24]. The goal of having these parameters is to avoid pessimistic threshold values, leading to decrease the performance of TMVS.

*ECCs.* Error correction codes are software-based approaches that do not require expensive and time consuming pre-processing steps. However, an ECC implementation usually requires significant memory overhead. For example, to generate a 128-bit key with a targeted key error rate $\leq 10^{-6}$, the BCH coding in [9] requires 26.9 raw response bits to generate 1 reliable bit if the raw response bits exhibit errors with $p_e = 0.15$, and it requires 3.68 raw PUF bits to generate 1 stable PUF bit if $p_e = 0.06$ [25]. Additionally, powerful codes such as BCH and Reed-Muller have very complex decoders, thus, high power consumption and number of cycles result in when the decoder is implemented on general-purpose processors. Therefore, hardware implementations are designed to reach as inexpensive, fast, power-efficient as possible error correction techniques [9]. This solution is expensive in terms of area, a cost that can be unacceptable for resource-constrained devices. TMVS have a very simple decoder compared to the aforementioned ECCs, where mostly XORing operations are used, thus, it cancels the need for expensive hardware implementations. Other codes such as the repetition codes are not complex as well, but they suffer from large size of PUF response and helper data. To mitigate various ECCs limitations, concatenated codes were later developed and used [9], [23], with the repetition code as the outer code.

*Repetition Code.* To implement a repetition code of length $n$, a random bit $b$ is repeated $n$ times into an $n$-bit string and then XORed with an $n$-bit PUF response to produce the helper data. During the regeneration phase, the random bit $b$ is recovered by XORing the helper data and the PUF response and then running a majority voting on the XORed result [22]. Similarly to TMVS, the repetition code has very low complexity compared to other ECCs, and only simple operations like XORing are used. Moreover, repetition codes are indeed more efficient than our method in terms of PUF error rate reduction. They can correct up to $\frac{n-1}{2}$ errors, which is possible in our method only if the enrolled pattern is identical to the chosen code word $S$, hence $d_H(R, S) = 0$, but this is not always the case. Our error correction capability depends on the Hamming distance between the selected SRAM pattern and the chosen code word during enrollment, it ranges between $N_{e,\max}$ (maximum number of possible errors) and $\frac{n-1}{2}$. Overall, repetition code has low complexity and excellent error correction capability. However, repetition codes suffer a high entropy loss [23]. For an $n$-bit PUF response, the entropy loss is $n-1$ and there is at most one bit of leftover entropy. TMVS can achieve much smaller entropy loss, we show this through a toy example, and we omit the theoretical proof due to lack of space. Given an unbiased memory ($p = \frac{1}{2}$), we fix a single code word $S_0 = 00000$ and thresholds $TH_{low} = 1$ and $TH_{high} = 4$, assumed to be public data. According to (5)and (6), only a PUF response that differs from $S_0$ with 0, 1, 4 or 5 bits is selected and used to derive a key bit. The total number of such response values is equal to $\binom{5}{0} + \binom{5}{1} + \binom{5}{4} + \binom{5}{5} = 12$. Considering these 12 possible patterns, an attacker needs to have $\log_2(12) = 3.58$ bits information about the raw SRAM data to guess the extracted key bit. Thus, the leftover entropy in our approach is equal to 3.58 for a single extracted key bit, which is about 3x improvement compared to repetition code. Furthermore, a tighter selection interval $[TH_{low}, TH_{high}]$ and longer code word provide higher entropy. In fact, concatenated codes described previously are not without high entropy loss risk since the repetition code constitutes their outer code [23].

The helper data generated in our method reveals less information about raw SRAM data, thus, higher leftover entropy values can be achieved. A helper data leaking information about raw SRAM data is undesirable in terms of key secrecy, however it is useful for better error correction because it provides more information to the decoder to recover the occurred errors. This trade-off justifies how TMVS improves the leftover entropy at the cost of worst error correction capability and more memory overhead. Our approach sacrifices a larger size of PUF response2, since we only select SRAM patterns that satisfy (5) or (6). Hence, when a large number of code words is used, less SRAM patterns are discarded, and vice versa. Thus, the size of helper data required to store code words is inversely proportional to the size of required raw SRAM bits. Besides code words, the memory addresses of selected patterns needs to be stored also as helper data, and this requires (key length x address length) bits.

*Spatial Majority Voting (SMV).* Koeberl*et al.* [17] developed a method to reduce PUF error rate based on majority voting, where only major bits (zeros or ones with the highest number) are used as PUF bits and undergo the majority voting decoder. The error correction capability $t$ of their scheme is $t = \frac{n-1}{4}$, and other techniques such as preselection are required as complement to extract a highly reliable key with error rate $10^{-6}$. As explained in the previous paragraph, our $t$ value ranges between $N_{e,\max}$ (maximum number of possible errors) and $\frac{n-1}{2}$. Thus, for $N_{e,\max}$ values larger than $\frac{n-1}{4}$, our approach achieves better error rates. For example, with $k = 17$, if the raw PUF has $p_e = 0.15$, the error rate after SMV becomes $5.6 \cdot 10^{-3}$ [17]. However, in our method, $p_e$ can be reduced to $1.53 \cdot 10^{-5}$ according to (13) with $k = 17$ and $N_{e,\max}(3.5\sigma_e)$, which is a 360x improvement. TMVS can even achieve smaller error-rate values with an order of $10^{-6}$, as shows the numerical example following (13). Therefore, no additional techniques would be required to extract a secret key.

## VII. Conclusion

In this paper, we propose a new PUF processing technique called Threshold-based Majority Voting Scheme (TMVS). In this approach, we determine two thresholds that allow for a simple majority voting decoder to regenerate the cor-

rect key bits by sacrificing more raw PUF bits. We prove theoretically that our method can reduce the raw BER of SRAM PUF responses to $10^{-6}$, even for high raw BER of 15%, alone without relying on complementary solutions. We analytically compare TMVS to other well-known techniques used to enhance SRAM PUF reliability. We found that our approach has smaller error correction capability and larger memory overhead compared to powerful ECCs, however, it leaks less information about the generated key, and it decreases significantly ECC complexity, resulting in a reduced system cost for applications where minimizing logic-cost is the primary design constraint. Additionally, our proposed method requires minimal effort for estimating some SRAM PUF characteristics including the worst-case BER, an acceptable requirement compared to other pre-processing methods with bit-specific expensive characterization efforts. In future versions of this approach, we intend on finding the optimal codebook construction, then evaluate TMVS experimentally on low-power and crystal-free single chip $\mu$icro motes (SC$\mu$M) [26]. We also aim at proving that TMVS is information theoretically secure, taking into consideration biased SRAMs scenario.

## ACKNOWLEDGMENT

This document is issued within the frame and for the purpose of the OpenSwarm project. This project has received funding from the European Union's Horizon Europe Framework Programme under Grant Agreement No. 101093046. Views and opinions expressed are however those of the author(s) only and the European Commission is not responsible for any use that may be made of the information it contains.

## REFERENCES

[1] J. W. Lee, D. Lim, B. Gassend *et al.*, "A Technique to Build a Secret Key in Integrated Circuits for Identification and Authentication Applications," in *IEEE Symposium on VLSI Circuits (VLSI)*, 2004.

[2] B. Gassend, D. Clarke, M. Van Dijk *et al.*, "Silicon Physical Random Functions," in *ACM Conference on Computer and Communications Security (CCS)*, 2002.

[3] S. S. Kumar, J. Guajardo, R. Maes *et al.*, "The Butterfly PUF Protecting IP on Every FPGA," in *IEEE Hardware-Oriented Security and Trust (HOST)*, 2008.

[4] R. Maes, P. Tuyls, and I. Verbauwhede, "Intrinsic PUFs from Flip-flops on Reconfigurable Devices," in *Benelux workshop on information and system security (WISSec)*, 2008.

[5] P. Simons, E. van der Sluis *et al.*, "Buskeeper PUFs, a Promising Alternative to D flip-flop PUFs," in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2012.

[6] J. Guajardo, S. S. Kumar *et al.*, "FPGA Intrinsic PUFs and Their Use for IP Protection," in *Cryptographic Hardware and Embed. Sys.*, 2007.

[7] Y. Li, C.-H. Hwang, T.-Y. Li *et al.*, "Process-variation Effect, Metal-gate Work-function Fluctuation, and Random-dopant Fluctuation in Emerging CMOS Technologies," *Transactions on Electron Devices*, vol. 57, 2009.

[8] S. K. Mathew, S. K. Satpathy, M. A. Anders *et al.*, "16.2 A 0.19 pJ/b PVT-variation-tolerant Hybrid Physically Unclonable Function Circuit for 100% Stable Secure Key Generation in 22nm CMOS," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014.

[9] C. Bösch, J. Guajardo *et al.*, "Efficient Helper Data Key Extractor on FPGAs," in *Cryptographic Hardware and Embedded Sys. (CHES)*, 2008.

[10] V. Van der Leest, B. Preneel, and E. Van der Sluis, "Soft Decision Error Correction for Compact Memory-Based PUFs Using a Single Enrollment," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. Springer, 2012.

[11] A. R. Korenda, F. Afghah, B. Cambou *et al.*, "A Proof of Concept SRAM-based Physically Unclonable Function (PUF) Key Generation Mechanism for IoT Devices," in *2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, 2019, pp. 1–8.

[12] M. Yue, "A Two-Stage TMV SRAM PUF Preselection Method with Fewer ECC Resources," in *2024 IEEE 7th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, vol. 7, 2024, pp. 528–533.

[13] M. Liu, C. Zhou, Q. Tang *et al.*, "A Data Remanence Based Approach to Generate 100% Stable Keys from an SRAM Physical Unclonable Function," in *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2017.

[14] W. Wang, A. D. Singh, and U. Guin, "A Systematic Bit Selection Method for Robust SRAM PUFs," *Journal of Electronic Testing*, vol. 38, no. 3, 2022.

[15] M. Gong, H. Zhang, C. Wang, Q. Tong, and Z. Liu, "Design and implementation of robust and low-cost SRAM PUF using PMOS and linear shift register extractor," *Microelectronics Journal*, vol. 103, p. 104844, 2020.

[16] R. Maes, P. Tuyls, and I. Verbauwhede, "Low-overhead Implementation of a Soft Decision Helper Data Algorithm for SRAM PUFs," in *Cryptographic Hardware and Embedded Sys. (CHES)*, 2009.

[17] P. Koeberl, J. Li, and W. Wu, "A Spatial Majority Voting Technique to Reduce Error Rate of Physically Unclonable Functions," in *International Conference on Trusted Systems (INTRUST)*, 2013.

[18] K. Liu, X. Chen, H. Pu *et al.*, "A 0.5-V Hybrid SRAM Physically Unclonable Function Using Hot Carrier Injection Burn-in for Stability Reinforcement," *Solid-State Circuits*, vol. 56, no. 7, 2020.

[19] S. Katzenbeisser, Ü. Kocabaş, V. Rožić *et al.*, "PUFs: Myth, Fact or Busted?" in *Cryptographic Hardware and Embedded Sys. (CHES)*, 2012.

[20] P. Koeberl, J. Li, R. Maes *et al.*, "Evaluation of a PUF Device Authentication Scheme on a Discrete 0.13 um SRAM," in *International Conference on Trusted Systems (INTRUST)*, 2012.

[21] S. Faour, B. Korecic, M. Vučinić, F. Maksimovic, D. C. Burnett, P. Muhlethaler, and T. Watteyne, "Single-Chip Motes and SRAM PUF: Feasibility Study," in *Workshop on Crystal-Free/-Less Radio and System-based Research for IoT (CrystalFreeIoT)*. IEEE, 2024.

[22] C. Böhm and M. Hofer, *Physical Unclonable Functions in Theory and Practice*. Springer Science & Business Media, 2012.

[23] P. Koeberl, J. Li, A. Rajan *et al.*, "Entropy Loss in PUF-based Key Generation Schemes: The Repetition Code Pitfall," in *IEEE Hardware-Oriented Security and Trust (HOST)*. IEEE, 2014.

[24] Y. Wang and M. Orshansky, "Efficient Helper Data Reduction in SRAM PUFs via Lossy Compression," in *IEEE Design, Automation & Test in Europe (DATE)*, 2018.

[25] J. Guajardo, S. S. Kumar, G.-J. Schrijen *et al.*, "Physical Unclonable Functions and Public-Key Crypto for FPGA IP Protection," in *IEEE Field Programmable Logic and Applications (FPL)*, 2007.

[26] F. Maksimovic, B. Wheeler, D. C. Burnett *et al.*, "A Crystal-Free Single-Chip Micro Mote with Integrated 802.15.4 Compatible Transceiver, sub-mW BLE Compatible Beacon Transmitter, and Cortex M0," in *2019 Symposium on VLSI Circuits*. IEEE, 2019, pp. C88–C89.