Two-Stage Threshold-based Majority Voting Scheme (TS-TMVS) for Robust SRAM PUFs

Sara Faour
sara.faour@inria.fr
Inria
Paris, France

Mališa Vučinić
malisa.vucinic@inria.fr
Inria
Paris, France

Thomas Watteyne
thomas.watteyne@inria.fr
Inria
Paris, France

Kristofer Pister
ksjp@berkeley.edu
University of California
Berkeley, CA, USA

ABSTRACT

Physically Unclonable Functions (PUFs) extract secret keys from intrinsic hardware, offering strong resistance to cloning and physical tampering. SRAM PUFs reuse memory at start-up but are sensitive to environmental noise. We present a two-stage scheme to extract reliable cryptographic keys from SRAM PUFs. The scheme extends our prior single-stage TMVS approach to reduce memory overhead while maintaining reliability. The method first extracts intermediate reliable bits from raw SRAM responses, then reapplies TMVS to achieve the target key error probability 10^{-6} with 128-bit keys. The two-stage cascade maintains all the benefits of the original design (low-complexity majority voting decoder, single SRAM measurement, low entropy loss) while reducing the SRAM memory overhead by 3.5× for 7.5% raw bit error rates. Our analysis reveals new error-memory trade-offs, demonstrating that the TS-TMVS maintains practical computational and memory requirements for resource-constrained devices.

CCS CONCEPTS

• **Security and privacy → Security in hardware**.

KEYWORDS

PUF, secret key, reliability, resource-constrained, Internet of Things.

1 INTRODUCTION

Physically Unclonable Functions (PUFs), such as SRAM PUFs, generate device-specific fingerprints through manufacturing variations. Their raw responses exhibit instability due to environmental factors and aging, which requires error correction for cryptographic applications. Traditional approaches employ fuzzy extractors with Error Correcting Codes (ECCs) to achieve key error probabilities below 10^{-6}, but face implementation challenges in resource-constrained devices such as Single-Chip μicro Motes (SCμM) [5, 12]. While powerful ECCs meet reliability targets, their computational complexity makes them impractical for many embedded systems. Simpler approaches like repetition codes rely on a simple majority voting decoder but incur excessive entropy loss. Recent concatenated coding schemes [8], with an inner code for initial error reduction and an outer code for final correction, offer a promising alternative by combining multiple coding layers to distribute computational complexity while maintaining reliability.

Our prior work [6] Threshold-based Majority Voting Scheme (TMVS) introduced an efficient approach based on the majority voting decoder in a novel way that avoids the pitfalls of standard repetition coding. The majority voting decoder is much simpler than a classical ECC decoder, consisting mainly of counting and threshold operations, which makes it efficient in software. TMVS achieved decoding simplicity, single-measurement operation, and low entropy loss, unlike traditional ECCs. TS-TMVS is a two-stage enhancement of TMVS that reduces memory overhead while preserving all the advantages of TMVS. The key innovation is to apply TMVS twice: first to raw SRAM bits to extract intermediate reliable bits, then a second time to achieve the target 128-bit key error probability of 10^{-6}. This cascaded implementation creates an error reduction pipeline that optimizes memory requirements while maintaining reliability. The contribution of this paper is threefold:

(1) We introduce TS-TMVS, a systematic two-stage adaptation of TMVS.
(2) We provide a theoretical analysis of error probability, memory requirements, and scalability of TS-TMVS.
(3) We analytically compare TS-TMVS with the single-stage approach, including theoretical derivations and qualitative evaluation of design trade-offs.

The remainder of this paper is organized as follows. Section 2 surveys related work. Section 3 provides a background on SRAM PUF and TMVS. Section 4 introduces TS-TMVS. Section 5 analytically evaluates the performance of TS-TMVS. Section 6 analytically compares TS-TMVS to TMVS. Finally, Section 7 concludes this work.

2 RELATED WORK

Reliable key generation from SRAM PUFs requires powerful error correction, yet resource-constrained devices demand low-complexity decoding. Conventional ECCs such as BCH and Reed-Muller [4] provide strong error resilience (targeting 10^{-6} key error rates), but incur high decoding complexity due to algebraic operations (e.g. finite-field arithmetic) and large helper data overhead [3]. Lighter codes such as Hamming codes [13] offer simpler decoders but insufficient error correction for typical SRAM PUF noise levels (5-25% BER).

To navigate the trade-off between error correction capability and implementation cost, several works have adopted hybrid coding schemes. Pointer-based methods such as Index-Based Syndrome coding reduce complexity to $O(n)$ by selecting reliable PUF bits via indices [2], although they incur entropy loss and struggle beyond 15% BER. Soft-decision helper data schemes [10] leverage probabilistic reliability metrics from multiple observations to achieve a near-optimal correction with $O(n \log n)$ complexity, but require 3-5× memory overhead. Concatenated codes [3, 7] use simple inner codes (e.g. repetition codes) to reduce the raw bit error rate, and combine with stronger outer codes (e.g. BCH, Golay, or Reed–Muller) to finalize error correction. Such concatenated schemes achieve high reliability with simpler decoding logic, offering a more efficient trade-off than single-layer ECCs. While repetition-based

constructions allow majority voting decoders, they incur significant entropy loss [9]. Hybrids combining temporal majority voting with ECCs [14] demonstrate efficient pre-processing (e.g. reducing raw BER to < 8%), but require entropy compensation mechanisms. More advanced variants leverage sequential helper data updates or machine learning-assisted [11] reliability prediction to dynamically optimize the correction-complexity trade-off.

Our proposed two-stage majority voting algorithm retains the lightweight nature of majority voting, but avoids repeated measurements (unlike soft decision schemes), and minimizes entropy loss (unlike repetition codes). Through its two-stage design, it balances between decoding simplicity, memory demands, and key reliability to offer a practical solution for microcontroller deployments.

3 PRELIMINARIES

3.1 SRAM PUF Fundamentals

A typical SRAM cell, such as the 6-transistor (6T) design, comprises six transistors ideally expected to behave identically. However, nanometer-scale process variations break this symmetry, producing unique and unpredictable characteristics per cell. On power-up, stable cells consistently initialize to a preferred state ('0' or '1'), while unstable ones are influenced by noise and environmental factors such as temperature and voltage. This randomness leads to variation across repeated power-ups. The Hamming distance $d_H(R, R')$ quantifies the number of differing bits between two responses R and R'. The raw bit error probability p_e – the likelihood of a bit flipping between instances – is critical to reliability, and is often minimized via stabilization techniques. Assuming that each bit arises from an independent and identically distributed (*i.i.d.*) process [1], the Hamming weight of an n-bit string, defined as the number of non-zero bits, follows a binomial distribution with success probability p, where each cell outputs '1' with probability p. The corresponding probability mass function is the probability of observing exactly i 1's in n independent bits, and is given by:

$$f_B(i; n, p) = \binom{n}{i} p^i (1-p)^{n-i}, \tag{1}$$

3.2 Threshold-based Majority Voting Scheme (TMVS)

We adopt a threshold-based selection approach called "TMVS" [6]. Assuming unbiased SRAM memory, where $p = \frac{1}{2}$, TMVS evaluates the Hamming distance between an n-bit SRAM response and candidate codewords from a predefined codebook. Responses whose distance is too close to $\frac{n}{2}$ for all codewords are discarded as they are more susceptible to bit-flip errors. The number of bit-flips n_e in an n-bit response follows a binomial distribution, which can be approximated by a normal distribution with mean $\mu_e = np_e$ and standard deviation $\sigma_e = \sqrt{np_e(1-p_e)}$. According to the empirical rule, nearly all errors lie within $[\mu_e \pm x_\sigma \sigma_e]$ for sufficiently large x_σ. Choosing $x_\sigma = 6$ bounds the error probability to $1.973 \cdot 10^{-9}$, and the worst-case number of bit flips is estimated as $N_{e,\max}(x_\sigma) = \lceil \mu_e + x_\sigma \cdot \sigma_e \rceil$. TMVS selects a sequence R only if there exists a codeword S such that the Hamming distance $d_H(R, S)$ deviates from np by at least $N_{e,\max}$, ensuring correct decoding via majority voting even under worst-case noise conditions.

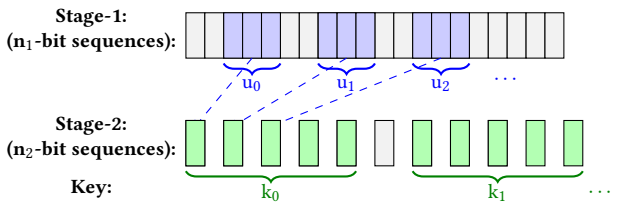


Figure 1: Two-stage SRAM key extraction example. Each n_1-bit sequence (u_i) maps to Stage-2 cells ($u_0 \rightarrow$1st, $u_1 \rightarrow$2nd, $u_2 \rightarrow$3rd). Green n_2-bit sequences form key bits (k_j).

Enrollment. A n-bit SRAM segment R is compared to each codeword S. If it satisfies the selection thresholds:

$$d_H(R, S) \leq \lfloor np \rfloor - N_{e,\max} \triangleq TH_{\text{low}}, \tag{2}$$

$$d_H(R, S) \geq \lceil np \rceil + N_{e,\max} \triangleq TH_{\text{high}}, \tag{3}$$

it is retained and its start address and codeword index are stored in the helper data. Otherwise, the SRAM scan continues from the sequence R starting at the next bit.

Regeneration. The helper data retrieves the codeword S assigned to the regenerated noisy sequence R'. The Hamming distance $d_H(R', S)$ is recalculated and the key bit is reconstructed using majority voting:

$$d_H(R', S) \leq \lfloor np \rfloor \rightarrow \text{bit key} = \text{'0'} \tag{4}$$

$$d_H(R', S) \geq \lceil np \rceil \rightarrow \text{bit key} = \text{'1'} \tag{5}$$

Codebook. The code C contains $M = |C|$ binary codewords of length n, parameterized by (n, x_σ). These parameters directly govern $N_{e,\max}$, which quantifies the selection thresholds and the error correction capability of the code. We assume unbiased SRAM with $p = 0.5$, therefore, we have symmetric thresholds (i.e. $TH_{\text{low}} = TH_{\text{high}}$). The codebook is constructed such that two codewords have a minimum Hamming distance of $d = 2 \cdot TH_{\text{low}} + 1$ and a maximum of $n - d$, ensuring that no SRAM response can simultaneously satisfy the threshold conditions (2) and (3) for more than one codeword. Due to this dual-distance constraint, conventional ECC codebooks are generally not suitable for TMVS. In our implementation, the codewords are generated by exhaustive search for $n < 14$ and by random sampling for $n \geq 15$, yielding a feasible set of M codewords under practical constraints. This generation of codebooks is not inherent to TMVS, and a more efficient generation is possible through structured methods. Since the codebook is fixed, it can be hard coded or stored in Read-Only Memory (ROM).

4 TWO-STAGE TMVS (TS-TMVS)

Although TMVS has low complexity, it requires a large number of SRAM bits. Based on experimental characterization of SCμM chips in [5], we estimate $p_e = 0.05$, at which TMVS requires over 5000 SRAM bits. To address this limitation, we propose a novel two-stage scheme that applies TMVS hierarchically. Using two short error-correcting codes in cascade, we reduce memory overhead while achieving the desired error probability. This scheme comprises the enrollment and regeneration phases.

Algorithm 1 Enroll Algorithm for TS-TMVS

Require: SRAM response $\mathbf{R} \in \{0,1\}^{N_{\text{SRAM}}}$, codes C_1, C_2, thresholds $(TH_{\text{low},1}, TH_{\text{high},1})$, $(TH_{\text{low},2}, TH_{\text{high},2})$
Ensure: Secret key $\mathbf{k} \in \{0,1\}^{n_k}$ and helper data $\mathcal{D} = (\mathcal{A}_1, \mathcal{S}_1, \mathcal{S}_2)$
Initialize $\mathbf{u}, \mathbf{k}, \mathcal{A}_1', \mathcal{S}_1', \mathcal{A}_1, \mathcal{S}_1, \mathcal{S}_2 \leftarrow \emptyset$
Stage-1: Intermediate Bit Extraction
for $i = 0$ to $N_{\text{SRAM}} - n_1$ **do**
  $\mathbf{R}_1 \leftarrow \mathbf{R}[i : i + n_1 - 1]$
  for all $\mathrm{S}_1 \in C_1$ **do**
    $d \leftarrow d_H(\mathbf{R}_1, \mathrm{S}_1)$
    if $d \le TH_{\text{low},1}$ **then**
      Append i, index of S_1, and 0 to $\mathcal{A}_1'$, $\mathcal{S}_1'$, and $\mathbf{u}$; **break**
    else if $d \ge TH_{\text{high},1}$ **then**
      Append i, index of S_1, and 1 to $\mathcal{A}_1'$, $\mathcal{S}_1'$, and $\mathbf{u}$; **break**
    end if
  end for
end for
Stage-2: Final Key Extraction
for $j = 0$ to $|\mathbf{u}| - n_2$ **do**
  $\mathbf{R}_2 \leftarrow \mathbf{u}[j : j + n_2 - 1]$
  for all $\mathrm{S}_2 \in C_2$ **do**
    $d \leftarrow d_H(\mathbf{R}_2, \mathrm{S}_2)$
    if $d \le TH_{\text{low},2}$ or $d \ge TH_{\text{high},2}$ **then**
      Append derived bit to $\mathbf{k}$ and index of S_2 to $\mathcal{S}_2$
      for $m = 0$ to $n_2 - 1$ **do**
        Append $\mathcal{A}_1'[j+m]$ to $\mathcal{A}_1$ and $\mathcal{S}_1'[j+m]$ to $\mathcal{S}_1$
      end for
      break
    end if
  end for
end for
return $(\mathbf{k}, \mathcal{D} = (\mathcal{A}_1, \mathcal{S}_1, \mathcal{S}_2))$

Algorithm 2 Regeneration Algorithm for TS-TMVS

Require: Noisy SRAM response $\mathbf{R}' \in \{0,1\}^{N_{\text{SRAM}}}$, helper data $\mathcal{D} = (\mathcal{A}_1, \mathcal{S}_1, \mathcal{S}_2)$, code lengths n_1, n_2
Ensure: Reconstructed key $\mathbf{k} \in \{0,1\}^{n_k}$
Initialize $\mathbf{u}, \mathbf{k} \leftarrow \emptyset$; $n_k \leftarrow |\mathcal{S}_2|$
Stage-1: Reconstruct Intermediate Bits
for $i = 0$ to $n_2 \cdot n_k - 1$ **do**
  $a_i \leftarrow \mathcal{A}_1[i]$, $s_i \leftarrow \mathcal{S}_1[i]$
  $\mathbf{R}_1 \leftarrow \mathbf{R}'[a_i : a_i + n_1 - 1]$
  $\mathrm{S}_1 \leftarrow$ codeword in C_1 with index s_i
  $d \leftarrow d_H(\mathbf{R}_1, \mathrm{S}_1)$
  if $d \le \left\lfloor \frac{n_1}{2} \right\rfloor$ **then**
    Append 0 to $\mathbf{u}$
  else
    Append 1 to $\mathbf{u}$
  end if
end for
Stage-2: Reconstruct Final Key
for $j = 0$ to $n_k - 1$ **do**
  $\mathbf{u}_j \leftarrow \mathbf{u}[j \cdot n_2 : (j+1) \cdot n_2 - 1]$
  $\mathrm{S}_2 \leftarrow$ codeword in C_2 with index $\mathcal{S}_2[j]$
  $d \leftarrow d_H(\mathbf{u}_j, \mathrm{S}_2)$
  if $d \le \left\lfloor \frac{n_2}{2} \right\rfloor$ **then**
    Append 0 to $\mathbf{k}$
  else
    Append 1 to $\mathbf{k}$
  end if
end for
return $\mathbf{k}$

4.1 Enrollment Procedure (Algorithm 1)

Codebooks. Let $C_1 \subseteq \{0,1\}^{n_1}$ and $C_2 \subseteq \{0,1\}^{n_2}$ be two ECCs with selection thresholds $(TH_{\text{low},1}, TH_{\text{high},1})$ and $(TH_{\text{low},2}, TH_{\text{high},2})$, respectively.

Stage-1 (Intermediate Bits). The SRAM of size N_{SRAM} (bits) is scanned with a sliding window of size n_1. Each sequence $\mathbf{R}_1$ is assigned a bit $b_i \in \{0,1\}$ if its Hamming distance to some codeword $\mathrm{S}_1 \in C_1$ is below $TH_{\text{low},1}$ (mapped to 0) or above $TH_{\text{high},1}$ (mapped to 1). This yields a sequence $\mathbf{u}$ of L_1 intermediate bits, with their addresses and matched codeword indices temporarily stored.

Stage-2 (Key Bits). A sliding window of size n_2 is applied to $\mathbf{u}$. Each sequence $\mathbf{R}_2$ is assigned a bit $k_j \in \{0,1\}$ if its Hamming distance to some codeword $\mathrm{S}_2 \in C_2$ is below $TH_{\text{low},2}$ (mapped to 0) or above $TH_{\text{high},2}$ (mapped to 1). Each accepted sequence yields a single key bit, forming the final key $\mathbf{k} \in \{0,1\}^{n_k}$, as shown in Fig. 1.

Helper Data. Only intermediate bits from Stage-1 used in accepted Stage-2 sequences are stored. The output helper data are $\mathcal{D} = (\mathcal{A}_1, \mathcal{S}_1, \mathcal{S}_2)$, where $\mathcal{A}_1$ and $\mathcal{S}_1$ hold the addresses and codeword indices of the $n_2 \cdot n_k$ retained Stage-1 segments, and $\mathcal{S}_2$ holds the n_k codeword indices of Stage-2. This public data is stored on programmable NVM memory such as Flash, or on external devices.

4.2 Regeneration Procedure (Algorithm 2)

Stage-1 (Intermediate Bits). Using the helper data $\mathcal{A}_1$ and $\mathcal{S}_1$, $n_2 \cdot n_k$ sequences of length n_1 each are extracted from the noisy SRAM response for each address $a_i \in \mathcal{A}_1$. Each sequence is compared to the corresponding codeword $\mathrm{S}_1 \in C_1$, indexed by $\mathcal{S}_1[i]$. If the Hamming distance is at most $\lfloor n_1/2 \rfloor$, the sequence is decoded to 0; otherwise, to 1. This yields a sequence $\mathbf{u} \in \{0,1\}^{n_2 \cdot n_k}$ of intermediate bits.

Stage-2 (Key Bits). The intermediate bit sequence $\mathbf{u}$ is divided into n_k segments of length n_2. Each segment is compared to a codeword $\mathrm{S}_2 \in C_2$, indexed by $\mathcal{S}_2[j]$. If the Hamming distance is at most $\lfloor n_2/2 \rfloor$, the segment is decoded to key bit 0; otherwise, to 1. The resulting sequence $\mathbf{k} \in \{0,1\}^{n_k}$ is the recovered key.

5 THEORETICAL ANALYSIS

Notation. In TS-TMVS, we use two codes: $C_1 \subseteq \{0,1\}^{n_1}$ and $C_2 \subseteq \{0,1\}^{n_2}$, with corresponding selection intervals defined as: $T_1 = [0, TH_{\text{low},1}] \cup [TH_{\text{high},1}, n_1]$, $T_2 = [0, TH_{\text{low},2}] \cup [TH_{\text{high},2}, n_2]$, and codebooks sizes $M_1 = |C_1|$ and $M_2 = |C_2|$. For single stage TMVS, we use the code $C \subseteq \{0,1\}^n$ with size $M = |C|$ and define a single selection interval: $T = [0, TH_{\text{low}}] \cup [TH_{\text{high}}, n]$.

5.1 Selection Probability

We define P_{select} the probability that an SRAM sequence R is selected and enrolled with respect to the codebook C, given by:

$$P_{\text{select}}(n,T) = M \cdot \sum_{d \in T} f_B(d;n,0.5) \tag{6}$$

Proof: See Appendix A.

5.2 Error Decoding Probability

5.2.1 Single-Stage Model. We define P_{error} as the probability for an enrolled key bit to be wrongly decoded using the majority voting decoder. A decoding error occurs when the noisy SRAM response of a key bit enrolled '0' does not satisfy (4), or when an enrolled '1' does not satisfy (5). This probability is given by:

$$\begin{aligned} &P_{\text{error}}(n,p_e,T) \\ &= \frac{M}{P_{\text{select}}} \sum_{d'=\lceil \frac{n}{2} \rceil}^{n} \sum_{d \in T} \Big[f_B(d;n,0.5) \cdot \sum_{n_X=0}^{z} \big[f_B(n_X;d,p_e) \\ &\qquad \cdot f_B(d'-d+n_X;n-d,p_e) \big] \Big] \end{aligned} \tag{7}$$

Proof: See Appendix B.

5.2.2 Two-Stage Model. The two-stage decoding probability compounds the errors of the two cascaded decoders. The Stage-1 decoder induces an intermediate bit error probability given by $P_{\text{error},1} = P_{\text{error}}(n_1,p_e,T_1)$, which serves as the effective noise rate for the Stage-2 decoder. This yields a final key bit error probability $P_{\text{error},2} = P_{\text{error}}(n_2,P_{\text{error},1},T_2)$, giving the full expression:

$$P_{\text{error, TS}} = P_{\text{error}}(n_2, P_{\text{error}}(n_1,p_e,T_1),T_2) \tag{8}$$

5.2.3 Key Failure probability. A key can only be successfully reconstructed when all the decoded key bits are correct. Thus, the failure probability is defined as follows.

$$P_{fail} = 1 - (1 - P_{error})^{n_k} \tag{9}$$

To achieve $P_{\text{fail}} \leq 10^{-6}$ for $n_k = 128$, a target bit error rate of $P_{\text{error}} \leq 7.81 \cdot 10^{-9}$ must be satisfied.

5.3 Memory Requirements

5.3.1 SRAM Requirement.

Single-Stage Model. To generate a key of length n_k using a single code C of length n, the number of SRAM source bits required during enrollment is:

$$N_{\text{SRAM}}(n_k,n,T) = n_k \left(n + \frac{1}{P_{\text{select}}(n,T)} - 1 \right) \tag{10}$$

Proof: See Appendix C.1.

Two-Stage Model. The Stage-2 code C_2 needs n_k accepted sequences of length n_2, requiring $N_{\text{SRAM},2} = N_{\text{SRAM}}(n_k,n_2,T_2)$ bits. To generate $N_{\text{SRAM},2}$ intermediate bits at Stage 1, the number of raw SRAM bits needed is $N_{\text{SRAM},1} = N_{\text{SRAM}}(N_{\text{SRAM},2},n_1,T_1)$. This total $N_{\text{SRAM},1}$ represents the full SRAM size needed at enrollment and can be written compactly as:

$$N_{\text{SRAM, TS}} = n_k \left(n_2 + \frac{1}{P_{\text{select}}(n_2,T_2)} - 1 \right) \left(n_1 + \frac{1}{P_{\text{select}}(n_1,T_1)} - 1 \right) \tag{11}$$

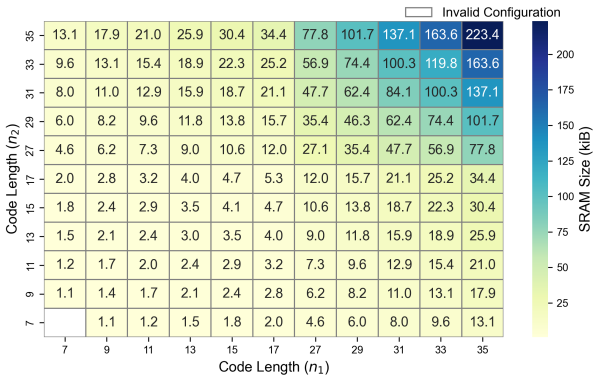


Figure 2: Minimum SRAM requirement (in kiB) for a two-stage configuration using codes of lengths n_1 and n_2 at $p_e = 0.05$. Invalid configuration has $P_{\text{fail}} > 10^{-6}$ for a 128-bit key.

5.3.2 Helper Data Requirement.

Single-Stage Model. Helper data stores selected SRAM addresses and the indices of their matched codewords. Assuming a pointer size of w bits, the helper data size is:

$$N_{\text{helper}} = 2w + (n_k - 1)\left\lceil \log_2(N_{\text{SRAM}}) \right\rceil + n_k \left\lceil \log_2(M) \right\rceil \tag{12}$$

Proof: See Appendix C.2.

Two-Stage Model. The size of the helper data defined in Section 4.1 as $\mathcal{D} = (\mathcal{A}_1, \mathcal{S}_1, \mathcal{S}_2)$ is:

$$\begin{aligned} N_{\text{helper, TS}} = {} & 3w + n_k n_2 \left\lceil \log_2(N_{\text{SRAM, TS}}) \right\rceil \\ & + n_k n_2 \left\lceil \log_2(M_1) \right\rceil + n_k \left\lceil \log_2(M_2) \right\rceil \end{aligned} \tag{13}$$

Proof: See Appendix D.
If the same code is reused for both stages, one codebook pointer can be omitted, reducing the size by w bits.

5.3.3 Codebook Size.

Single-Stage Model. The number of bits needed to store a codebook is $N_{\text{code}} = n \cdot M$.

Two-Stage Model. If the used codes are different, their codebooks must be stored independently. Thus, the total memory of the codebook is $N_{\text{code, TS}} = n_2 M_2 + \delta \cdot n_1 M_1$, where $\delta = 1$ if $C_1 \neq C_2$ (or their parameters differ) and $\delta = 0$ otherwise.

5.3.4 Resulting TS-TMVS Trade-offs. We now analyze the resulting trade-offs of TS-TMVS by evaluating the previously derived theoretical expressions across various configurations. As an online addition to this paper, the code used to analyze TS-TMVS is available on GitHub[1].

Code Length Impact. Fig. 2 shows the SRAM size required to meet the target error rate $P_{\text{error, TS}} \leq 7.8 \cdot 10^{-9}$ (corresponding to $P_{\text{fail}} \leq 10^{-6}$ for a 128-bit key) for different combinations (n_1, n_2), with fixed flip probability $p_e = 0.05$. Configurations exceeding the target error rate are marked invalid, with $(7, 7)$ being the only such case due to the limited correction capability. We observe a

[1] https://github.com/Sara-Fa/SRAM-PUF-key-generation/tree/main/two_stage_tmvs

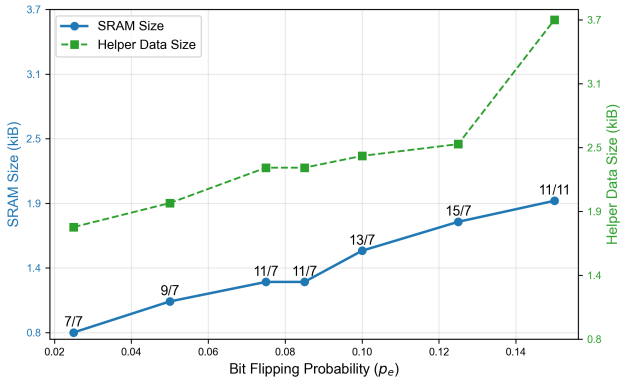


Figure 3: Optimal SRAM and helper data sizes across p_e for configurations satisfying $P_{\text{fail}} \leq 10^{-6}$ for a 128-bit key. Labels show code lengths (n_1/n_2) used at (Stage-1/Stage-2).

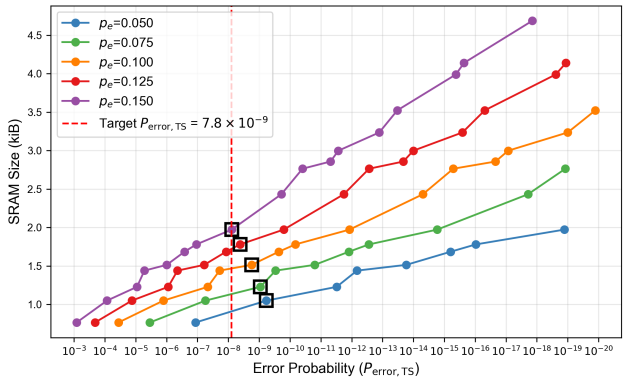


Figure 4: SRAM-error trade-off for different p_e values.

symmetric structure due to the interchangeability of the Stage-1 and Stage-2 roles. SRAM size increases monotonically with the code length, driven by the sharp drop in the selection probability for longer codes.

Resource Trajectory Analysis. Fig. 3 shows the minimum required SRAM and helper data sizes (with $w = 32$ for 32-bit systems) as functions of the bit flipping probability p_e, across all code combinations achieving the target key failure error $P_{\text{fail}} \leq 10^{-6}$ for a 128-bit key. As expected, SRAM size increases with p_e due to the need to select more robust (longer) codes with larger lengths and tighter selection thresholds, which inherently reduce the selection probability P_{select} and thus increase total SRAM consumption, as captured by (10). The helper data size exhibits a generally increasing trend, but its behavior depends on both the code lengths and their assignment to the two stages. According to (13), the dominant term scales linearly with $n_k n_2$, where n_k is constant, while other components such as $\log_2(N_{\text{SRAM, TS}})$, $\log_2(M_1)$, and $\log_2(M_2)$ contribute only logarithmically. Therefore, to minimize helper data size, it is always preferable to reduce n_2 – i.e., assign shorter codes to Stage-2. This choice reduces the number of retained intermediate bits and reduces both address offset and Stage-1 codeword index terms. Shorter codes used in Stage-2 also benefit from a higher selection probability, which reduces the number of discarded intermediate bits needed to produce a fixed number of output key bits. Overall, configurations with longer codes in Stage 1 and shorter ones in Stage-2 consistently achieve better helper data efficiency.

Error-SRAM Trade-off Across Noise Levels. Each curve traces the Pareto-optimal configurations minimizing SRAM usage for a given decoding error. Lower p_e values yield flatter frontiers, while higher p_e values require more SRAM to maintain low error rates. The red dashed line indicates the decoding error threshold $7.8 \cdot 10^{-9}$, which corresponds to $P_{\text{fail}} = 10^{-6}$ for a 128-bit key. The black square markers highlight the first configuration along each Pareto frontier (corresponding to different values of p_e) that achieves this target error with the smallest possible SRAM footprint.

6 COMPARING TO TMVS

Using the theoretical results from Section 5, we compare TS-TMVS' overhead with the single-stage approach and briefly explain in Appendix E why adding a third stage is suboptimal.

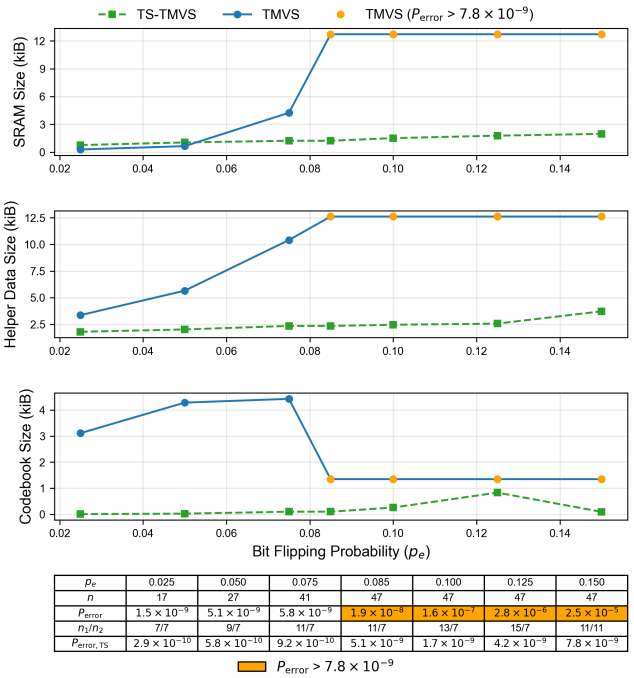
6.1 Memory Overhead

Fig. 5 compares the memory demands of the two-stage and single-stage methods in terms of SRAM, helper data, and codebook size over increasing bit-flip probabilities p_e. At low noise levels (e.g., $p_e \leq 0.05$), which are less common in real SRAM PUFs, the single-stage TMVS requires slightly less SRAM. However, when p_e increases to 0.075, its SRAM usage grows to 4.2 kiB compared to 1.2 kiB for TS-TMVS– a 3.5x reduction in memory requirements. At $p_e = 0.085$, TMVS' efficiency degrades rapidly and it fails to meet the target error $P_{\text{error}} \leq 7.8 \cdot 10^{-9}$ even with the largest tested code (length 47), implying even larger codes–and higher memory costs–would be needed. In contrast, the two-stage method scales smoothly with p_e and reliably achieves the error constraint with significantly lower SRAM usage under high noise. The helper data subplot shows that the two-stage method consistently requires less memory for all p_e, due to its partitioned structure that limits retained segments and reduces pointer/index overhead, as discussed in Section 5.3.4. Codebook size trends are less uniform, but generally favor the two-stage method, which uses shorter codes, and hence smaller individual codebooks. Codebooks are pre-generated externally, stored in ROM, and require no on-device computation. Overall, combining reductions in SRAM and helper data makes the two-stage method more scalable and practical for noisy and resource-constrained settings. The accompanying table reports the code lengths and decoding error probabilities across p_e values (column headers). Orange cells mark single-stage configurations that exceed the error threshold.

***Remark*:** We assume a 32-bit platform with pointer size $w = 32$, typical of many constrained microcontrollers.

6.2 Time and Space Complexity

We focus our analysis on the regeneration phase, as it occurs each time the secret key is reconstructed, unlike enrollment which is performed only once per device. In TMVS, each of the n_k secret bits



p_e	0.025	0.050	0.075	0.085	0.100	0.125	0.150
n	17	27	41	47	47	47	47
P_{error}	1.5×10^{-9}	5.1×10^{-9}	5.8×10^{-9}	1.9×10^{-8}	1.6×10^{-7}	2.8×10^{-6}	2.5×10^{-5}
n_1/n_2	7/7	9/7	11/7	11/7	13/7	15/7	11/11
$P_{error,TS}$	2.9×10^{-10}	5.8×10^{-10}	9.2×10^{-10}	5.1×10^{-9}	1.7×10^{-9}	4.2×10^{-9}	7.8×10^{-9}

$P_{error} > 7.8 \times 10^{-9}$

Figure 5: Resource usage vs. p_e (with $P_{error} \leq 7.8 \cdot 10^{-9}$, corresponding to $P_{fail} \leq 10^{-6}$ for a 128-bit key).

is derived by reading a noisy SRAM segment R' and its corresponding codeword S, both of length n, using the addresses and codeword indices stored in the helper data; the two are then compared via bitwise XOR, summation (to compute Hamming distance), and a comparison against $\frac{n}{2}$. These operations are repeated n_k times on sequences of length n, and if $n > 32$, devices with 32-bit microprocessor must process each sequence in multiple 32-bit steps.

The two-stage method first performs similar operations on $n_k \cdot n_2$ segments of length n_1, producing $n_k \cdot n_2$ intermediate bits stored in registers. Then, a second round of n_k XOR-sum-compare steps is applied to groups of n_2 intermediate bits. Since our results show that $n_2 \leq 11$, the total number of operations in the two-stage method—approximately $(n_2+1) \cdot n_k$—is up to 11 times higher than in the single-stage case, assuming the latter uses a code length $n \leq 32$. However, because both n_1 and n_2 are small, all operations in the two-stage scheme fit within a single 32-bit register. The intermediate $n_2 \cdot n_k$ bits produced by Stage-1 must also be stored temporarily, unlike in the single-stage method. Importantly, n_2 is chosen to be smaller than n_1 to minimize this overhead. In contrast, for single-stage codes with $n > 32$, operations must be split across multiple CPU cycles, reducing the time complexity advantage. Overall, the added complexity of the two-stage approach remains manageable, and the real-world performance gap between the two schemes is relatively small.

7 CONCLUSION

We present the Two-Stage TMVS (TS-TMVS) algorithm that reduces the memory overhead by 3.5× at 7.5% bit-flip rate while preserving the low-complexity advantage of the original single-stage TMVS approach. Through comprehensive analysis of SRAM, helper data, and codebook requirements, we demonstrate superior efficiency over the single-stage approach, especially at higher error rates where TMVS does not achieve the target reliability 10^{-6} with practical resources. This optimized error-memory trade-off enables robust cryptographic key generation for resource-constrained devices.

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] Frederik Armknecht, Roel Maes, Ahmad-Reza Sadeghi, François-Xavier Standaert, and Christian Wachsmann. 2011. A Formal Foundation for the Security Features of Physical Functions. In *IEEE Symposium on Security and Privacy*. IEEE, Oakland, CA, USA, 397–412.

[2] Georg T Becker, Alexander Wild, and Tim Güneysu. 2015. Security analysis of index-based syndrome coding for PUF-based key generation. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, Washington, DC, USA, 20–25.

[3] Christoph Bösch, Jorge Guajardo, Ahmad-Reza Sadeghi, Jamshid Shokrollahi, and Pim Tuyls. 2008. Efficient Helper Data Key Extractor on FPGAs. In *Cryptographic Hardware and Embedded Systems (CHES)*. Springer, Berlin, Heidelberg, 181–197.

[4] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. 2004. Fuzzy extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. In *Advances In Cryptology-EUROCRYPT 2004: International Conference On The Theory And Applications Of Cryptographic Techniques*. Springer, Berlin, Heidelberg, 523–540.

[5] Sara Faour, Blaz Korecic, Mališa Vučinić, Filip Maksimovic, David C. Burnett, Paul Muhlethaler, and Thomas Watteyne. 2024. Single-Chip Motes and SRAM PUF: Feasibility Study. In *Workshop on Crystal-Free/-Less Radio and System-based Research for IoT (CrystalFreeIoT)*. IEEE, Hong Kong, 24–29.

[6] Sara Faour, Mališa Vučinić, Filip Maksimovic, David C. Burnett, Paul Muhlethaler, Thomas Watteyne, and Kristofer Pister. 2024. TMVS: Threshold-based Majority Voting Scheme for Robust SRAM PUFs. In *IEEE Symposium on Computers and Communications (ISCC)*. IEEE, Paris, France, 1–8.

[7] G David Forney. 1965. Concatenated codes. (1965).

[8] Matthias Hiller, Ludwig Kürzinger, and Georg Sigl. 2020. Review of error correction for PUFs and evaluation on state-of-the-art FPGAs. *Journal of cryptographic engineering* 20 (2020), 229–247.

[9] Patrick Koeberl, Jiangtao Li, Anand Rajan, and Wei Wu. 2014. Entropy Loss in PUF-based Key Generation Schemes: The Repetition Code Pitfall. In *IEEE Hardware-Oriented Security and Trust (HOST)*. IEEE, Arlington, VA, USA, 44–49.

[10] Lieneke Kusters and Frans MJ Willems. 2021. Multiple observations for secret-key binding with SRAM PUFs. *Entropy* 23 (2021).

[11] Vishalini R Laguduva, Srinivas Katkoori, and Robert Karam. 2020. Machine learning attacks and countermeasures for PUF-based IoT edge node security. *SN Computer Science* (2020), 670–675.

[12] Filip Maksimovic, Brad Wheeler, David C Burnett, et al. 2019. A Crystal-Free Single-Chip Micro Mote with Integrated 802.15.4 Compatible Transceiver, sub-mW BLE Compatible Beacon Transmitter, and Cortex M0. In *Symposium on VLSI Circuits*. IEEE, Kyoto, Japan, C88–C89.

[13] Hoang-Long Pham, Duy-Hieu Bui, Xuan-Tu Tran, and Orazio Aiello. 2024. SRAM-based Physically Unclonable Function using Lightweight Hamming-Code Fuzzy Extractor for Energy Harvesting Beat Sensors. In *2024 International Conference on Advanced Technologies for Communications (ATC)*. IEEE, Ho Chi Minh City, Vietnam, 499–504.

[14] Andrés Santana-Andreo, Pablo Saraza-Canflanca, Héctor Carrasco-Lopez, Piedad Brox, Rafael Castro-López, Elisenda Roca, and Francisco V Fernández. 2022. A DRV-based bit selection method for SRAM PUF key generation and its impact on ECCs. *Integration* 85 (2022), 1–9.

APPENDICES

A PROOF OF SELECTION PROBABILITY FORMULA (6)

The probability for any SRAM sequence R to be selected with respect to a single codeword $S \in C$, is, in fact, the probability for $d_H(R,S)$ to satisfy (2) or (3), which means that a sequence has at most TH_{low} or at least TH_{high} different bits with S. Assuming the probability of one bit of R having a '1' value is $p = 0.5$, then the probability of R to have a Hamming distance d to a fixed codeword $S = s$ follows a binomial distribution Binomial$(n, 0.5)$. Therefore, the probability for R of having a Hamming distance of at most TH_{low} or at least TH_{high} is given by:

$$P_{\text{select}}(s) = \sum_{d \in T} f_B(d; n, 0.5), \tag{14}$$

where $T = [0, TH_{\text{low}}] \cup [TH_{\text{high}}, n]$ is the selection interval.
We explained in Section 3 that the codewords do not overlap, i.e. an SRAM sequence R can be selected with respect to a single codeword only. We extend the proof of (14) from a fixed codeword $S = s$ to *some* codeword $S \in C$. Thus, the events of selection with respect to M different codewords are disjoint and the probability of their union (the event that a codeword is assigned to R) is equal to the sum of their probabilities. Therefore, since $P_{\text{select}}(s)$ is independent of s (owing to the assumption $p = 0.5$), the probability of selection with respect to C is equal to $M \cdot P_{\text{select}}(s)$, which combined with (14) implies (6).

B PROOF OF ERROR PROBABILITY FORMULA (7)

We define P_{error} as the probability for a single enrolled key bit, denoted K_i, to be wrongly decoded using the majority voting decoder, i.e. $K_i' \neq K_i$, where K_i' is the regenerated - decoded - key bit. Referring to the decoding rules presented in (4) and (5), a key bit '0' is wrongly decoded if and only if the Hamming distance between the noisy SRAM response R' and the codeword S that was assigned to it, is strictly larger than $\lfloor np \rfloor$, and vice versa when a key bit '1' is wrongly decoded. In the following proof, we denote by capital letters such as K_i and K_i' the random variables (RVs), and by small letters their values; for better clarity we use the notation P_Y to refer to the probability distribution of a RV Y. The definition of error probability given above can be represented as:

$$\begin{aligned}
P_{\text{error}} =& P_{K_i'K_i}(k_i' \neq k) \\
=& P_{K_i'K_i}(k_i' = 1 | k_i = 0) P_{K_i}(k_i = 0) \\
&+ P_{K_i'K_i}(k_i' = 0 | k_i = 1) P_{K_i}(k_i = 1) && (15) \\
=& P_{D'K_i}(d' \geq \lceil np \rceil | k_i = 0) P_{K_i}(k_i = 0) && (16) \\
&+ P_{D'K_i}(d' \leq \lfloor np \rfloor | k_i = 1) P_{K_i}(k_i = 1) && (17)
\end{aligned}$$

The remainder of this proof focuses on deriving the probability mass functions present in (16) and (17). First, we calculate $P_{K_i}(k_i = 0)$ and deduce $P_{K_i}(k_i = 1)$. Then we derive $P_{D'K_i}(d' \geq \lceil np \rceil | k_i = 0)$ and deduce $P_{D'K_i}(d' \leq \lfloor np \rfloor | k_i = 1)$ to avoid repetition.

We define the probability that an enrolled key bit has a value $k_i = 0$, as the probability of satisfying $d_H(R,S) \leq TH_{\text{low}}$ so that the sequence R is selected or enrolled with respect to a codeword S according to (2) and (3). This probability is derived using the probability mass function (pmf) of the binomial distribution given in (1), and assuming $p = 0.5$ as follows:

$$\begin{aligned}
P_{K_i}(k_i = 0) &= \frac{\sum_{l=0}^{TH_{\text{low}}} f_B(l; n, 0.5)}{\sum_{j \in T} f_B(j; n, 0.5)} && (18) \\
&= \frac{M}{P_{\text{select}}} \cdot \sum_{l=0}^{TH_{\text{low}}} f_B(l; n, 0.5), && (19)
\end{aligned}$$

where $T = [0, TH_{\text{low}}] \cup [TH_{\text{high}}, n]$ is the allowable distance interval, and the denominator in (18) is substituted using (6) for simplicity. Similarly, the probability for an enrolled key bit to have value $k_i = 1$ can be deduced using:

$$P_{K_i}(k_i = 1) = 1 - P_{K_i}(k_i = 0) \tag{20}$$

We define the RV $D' = d_H(R', S)$ as the Hamming distance of the regenerated SRAM response that cannot be correctly decoded. As described earlier, the values taken by D' are larger than or equal to $\lceil \frac{n}{2} \rceil$, i.e. $d' \in \{\lceil \frac{n}{2} \rceil, \ldots, n\}$, and since these events of D' are disjoint, we have the following equality:

$$P_{D'K_i}(d' \geq \lceil \frac{n}{2} \rceil | k_i = 0) = \sum_{d' = \lceil \frac{n}{2} \rceil}^{n} P_{D'K_i}(d' | k_i = 0) \tag{21}$$

Next, we calculate the probability of D' for having a value d'. We use the marginal probability distribution of $P_{D'K_i}(d' | k_i = 0)$, expressed in (23), where we introduce a new RV $D = d_H(R,S)$ defined as the Hamming distance obtained during enrollment of the SRAM sequence R with S. Given that the enrolled key bit has value $k_i = 0$ - since $P_{D'K_i}(d' | k_i = 0)$ is the conditional probability of D' given K_i - we can deduce that D takes value $d \in \{0, TH_{\text{low}}\}$.

$$\begin{aligned}
P_{D'K_i}(d' | k_i = 0) &= \sum_{d=0}^{TH_{\text{low}}} P_{D'DK_i}(d', d | k_i = 0) && (22) \\
&= \sum_{d=0}^{TH_{\text{low}}} \left[P_{D'DK_i}(d' | d, k_i = 0) \cdot P_{DK_i}(d | k_i = 0) \right] && (23)
\end{aligned}$$

The probability of D for having a specific value d with a sequence R, such that R is enrolled as $k_i = 0$, is given by:

$$P_{DK_i}(d | k_i = 0) = \frac{f_B(d; n, 0.5)}{\sum_{l=0}^{TH_{\text{low}}} f_B(l; n, 0.5)} \tag{24}$$

Now, we need to calculate $P_{D'DK_i}(d' | d, k_i = 0)$: the probability for a noisy SRAM response R' to have $d_H(R', S) = d'$ knowing that $d_H(R,S) = d$ and $k_i = 0$. We define the set $\mathcal{X} = \{j | (R \oplus S)_j = 1\}$ to describe the positions j of the bits that differ between R and S, i.e. where the j-th bit of $R \oplus S$ is 1. The set complement $\mathcal{X}^c$ is the set of all bit positions of R that are in the universal set $\{1, 2, \ldots, n\}$ but not in $\mathcal{X}$. The number of elements in each set are given by $|\mathcal{X}| = d$ and $|\mathcal{X}^c| = n - d$. We distinguish between two types of bit flips that occur during the generation of R': 1) bit flip at position $j \in \mathcal{X}$, resulting in a reduction in the number of different bits with S and thus decreasing d'; 2) bit flip at position $j \in \mathcal{X}^c$, resulting in an increase in the number of different bits with S and thus increasing d'. Recall that d' represents the value of $d_H(R', S)$ that is wrongly decoded, hence $d' > d$. We define the RV $N_{\mathcal{X}}$ as the number of bit flips occurring in R at positions $\{j_0, \ldots, j_{N_{\mathcal{X}}}\} \subseteq \mathcal{X}$. To obtain

d' from a reference value d, we have several scenarios: $n_{\mathcal{X}} = 0$ (no bit flips in positions $\mathcal{X}$) and $d' - d$ bit flips in $\mathcal{X}^c$ positions; $n_{\mathcal{X}} = 1$ and $d' - d + 1$ bit flips in $\mathcal{X}^c$; $n_{\mathcal{X}} = 2$ and $d' - d + 2$ bit flips in $\mathcal{X}^c$; etc. Thus, $n_{\mathcal{X}}$ bit flips can occur in positions $\mathcal{X}$ with probability $P_{N_{\mathcal{X}}DK_i}(n_{\mathcal{X}}|d, k_i = 0) = f_B(n_{\mathcal{X}}; d, p_e)$. To get a final Hamming distance d', given these $n_{\mathcal{X}}$ bit flips occurring in $\mathcal{X}$, $n_{\mathcal{X}}$ bit flips should occur in $\mathcal{X}^c$ to compensate the decrease of distance, plus $d' - d$ more bit flips in $\mathcal{X}^c$ to reach a final distance d', this event has a probability $P_{D'N_{\mathcal{X}}DK_i}(d'|n_{\mathcal{X}}, d, k_i = 0) = f_B(d' - d + n_{\mathcal{X}}; n - d, p_e)$. Therefore, we calculate $P_{D'DK_i}(d'|d, k_i = 0)$ using its marginal distribution - see (25) and (26) - to introduce the different $n_{\mathcal{X}}$ scenarios, and we substitute in (26) the pmfs $P_{N_{\mathcal{X}}DK_i}$ and $P_{D'N_{\mathcal{X}}DK_i}$ given above, as follows:

$$P_{D'DK_i}(d'|d, k_i = 0) = \sum_{n_{\mathcal{X}}=0}^{z} P_{D'N_{\mathcal{X}}DK_i}(d', n_{\mathcal{X}}|d, k_i = 0) \tag{25}$$

$$= \sum_{n_{\mathcal{X}}=0}^{z} [P_{D'N_{\mathcal{X}}DK_i}(d'|n_{\mathcal{X}}, d, k_i = 0) \cdot P_{N_{\mathcal{X}}DK_i}(n_{\mathcal{X}}|d, k_i = 0)] \tag{26}$$

$$= \sum_{n_{\mathcal{X}}=0}^{z} [f_B(d' - d + n_{\mathcal{X}}; n - d, p_e) \cdot f_B(n_{\mathcal{X}}; d, p_e)] \tag{27}$$

where the upper bound z of $n_{\mathcal{X}}$ follows from the two facts: 1) we can have at most d bit flips at positions $\mathcal{X}$, hence $n_{\mathcal{X}} \leq d$; 2) we can have at most $n-d$ bit flips at positions $\mathcal{X}^c$, hence $d' - d + n_{\mathcal{X}} \leq n - d$. By summing both inequalities, we obtain:

$$n_{\mathcal{X}} \leq \frac{n - d' + d}{2} \triangleq z \tag{28}$$

By replacing (24) in (23), and (23) in (21), we obtain:

$$P_{D'K_i}(d' \geq \lceil \frac{n}{2} \rceil | k_i = 0) = \sum_{d'=\lceil \frac{n}{2} \rceil}^{n} \sum_{d=0}^{TH_{\text{low}}} \left[\frac{f_B(d; n, 0.5)}{\sum_{i=0}^{TH_{\text{low}}} f_B(i; n, 0.5)} \cdot P_{D'DK_i}(d'|d, k_i = 0) \right] \tag{29}$$

When multiplying (19) by (29) the denominator term $\sum_{i=0}^{TH_{\text{low}}} f_B(i; n, 0.5)$ can be taken outside the summation symbols in (29), because it is independent of d and d', hence, it is simplified with the same numerator term in (19) to produce:

$$P_{D'K_i}(d' \geq \lceil \frac{n}{2} \rceil | k_i = 0) P_{K_i}(k_i = 0) = \frac{M}{P_{\text{select}}} \sum_{d'=\lceil \frac{n}{2} \rceil}^{n} \sum_{d=0}^{TH_{\text{low}}} [f_B(d; n, 0.5) \cdot P_{D'DK_i}(d'|d, k_i = 0)] \tag{30}$$

Finally, substituting (27) in (30) gives the term (16). To deduce the term (17), we replace TH_{low} by $n - TH_{\text{high}}$ and refer to $\overline{S}$ (complementary to S) in calculating the Hamming distance instead of S. This gives us:

$$P_{\text{error}} = \frac{M}{P_{\text{select}}} \sum_{d'=\lceil \frac{n}{2} \rceil}^{n} \sum_{d=0}^{\text{TH}_{\text{low}}} [f_B(d; n, 0.5) \cdot \sum_{n_{\mathcal{X}}=0}^{z} [f_B(n_{\mathcal{X}}; d, p_e) \cdot f_B(d' - d + n_{\mathcal{X}}; n - d, p_e)]] + \frac{M}{P_{\text{select}}} \sum_{d'=\lceil \frac{n}{2} \rceil}^{n} \sum_{d=0}^{n-\text{TH}_{\text{high}}} [f_B(d; n, 0.5) \cdot \sum_{n_{\mathcal{X}}=0}^{z} [f_B(n_{\mathcal{X}}; d, p_e) \cdot f_B(d' - d + n_{\mathcal{X}}; n - d, p_e)]] \tag{31}$$

Since the terms inside the summations are identical, we use the interval $T = [0, TH_{\text{low}}] \cup [TH_{\text{high}}, n]$ to simplify the two summation in (31) into a single one, which implies (7). This concludes the proof.

C PROOF OF TMVS MEMORY REQUIREMENTS

C.1 Proof of SRAM Requirement

During enrollment, SRAM sequences that do not meet the selection criteria (2) and (3) are skipped. Let N_{skipped} be the number of such skipped sequences when enrolling n_k bits as secret key. In total, $N_{\text{skipped}} + n_k$ SRAM sequences are tested with (2) and (3) to enroll n_k bits. Therefore, we represent P_{select}, calculated in Section 5.1, in a ratio format as follows:

$$P_{\text{select}} = \frac{n_k}{N_{\text{skipped}} + n_k} \tag{32}$$

Equation (32) allows us to deduce the ratio of skipped bits per single enrolled key bit:

$$\frac{N_{\text{skipped}}}{n_k} = \frac{1}{P_{\text{select}}} - 1 \tag{33}$$

If a selected sequence consumes n bits and a skipped one costs 1 bit (due to window sliding), the total number of required SRAM bits is:

$$N_{\text{SRAM}} = n_k \left(n + \frac{N_{\text{skipped}}}{n_k} \right) \tag{34}$$

Substituting (33) into (34), we obtain the SRAM requirement in (10):

$$N_{\text{SRAM}} = n_k \left(n + \frac{1}{P_{\text{select}}} - 1 \right) \tag{35}$$

C.2 Proof of Helper Data Requirement

Address Offsets: We store w the address of the first selected SRAM sequence (as a pointer), i.e. base address, and the offsets for the other $n_k - 1$ selected sequences. The number of bits required to store these offset values is at least $(n_k - 1)\lceil \log_2(N_{\text{SRAM}}) \rceil$. Thus, the number of bits required to store helper data related to SRAM addresses is:

$$N_{\text{helper},\mathcal{A}} = w + (n_k - 1) \left\lceil \log_2(N_{\text{SRAM}}) \right\rceil \tag{36}$$

Codeword Indices: Similarly, we store the address of the codebook C (only first element), and then the indices for the assigned codewords. Hence, the number of bits required to store the offsets is at least $n_k \lceil \log_2(M) \rceil$. Therefore, helper data size only increases logarithmically with an increasing number of required SRAM bits and codewords. The number of bits required to store helper data related to assigned codewords is defined as:

$$N_{\text{helper},\mathcal{S}} = w + n_k \left\lceil \log_2(M) \right\rceil \tag{37}$$

Total Helper Data: Adding both terms:

$$N_{\text{helper}} = N_{\text{helper},\mathcal{A}} + N_{\text{helper},\mathcal{S}} \tag{38}$$

$$= 2w + (n_k - 1)\left\lceil \log_2(N_{\text{SRAM}}) \right\rceil + n_k \left\lceil \log_2(M) \right\rceil \tag{39}$$

D PROOF OF TS-TMVS HELPER DATA REQUIREMENT

In the two-stage construction, only a subset of intermediate bits from Stage 1 – those used in accepted Stage-2 groups – are stored. Therefore, the helper data consists of:

$$\mathcal{D} = (\mathcal{A}_1, \mathcal{S}_1, \mathcal{S}_2)$$

Here:

- $\mathcal{A}_1$: base address and offsets for the $n_k \cdot n_2$ selected Stage-1 segments.
- $\mathcal{S}_1$: indices of the corresponding Stage-1 codewords.
- $\mathcal{S}_2$: indices of the Stage-2 codewords (one per key bit).

Each component requires:

$$N_{\mathcal{A}_1} = w + (n_k n_2 - 1)\left\lceil \log_2(N_{\text{SRAM, TS}}) \right\rceil$$

$$N_{\mathcal{S}_1} = w + n_k n_2 \left\lceil \log_2(M_1) \right\rceil$$

$$N_{\mathcal{S}_2} = w + n_k \left\lceil \log_2(M_2) \right\rceil$$

Adding all terms and simplifying using $(n_k n_2 - 1) \approx n_k n_2$, we obtain:

$$N_{\text{helper, TS}} = 3w + n_k n_2 \left\lceil \log_2(N_{\text{SRAM, TS}}) \right\rceil + n_k n_2 \left\lceil \log_2(M_1) \right\rceil + n_k \left\lceil \log_2(M_2) \right\rceil \tag{40}$$

If the same code is reused for both stages ($C_1 = C_2$), one pointer can be omitted, reducing the size by w bits.

E ADDING A THIRD-STAGE

To evaluate the optimal number of stages in the cascade architecture, we analyze the total SRAM bit requirement given by (11). It has been previously established (see Fig. 2) that shorter codes tend to minimize the SRAM size due to their higher selection probability. To provide a concrete illustration, we consider the smallest [2] usable code $n_1 = 7$ with perfect selection probability $P_{\text{select}}(n_1, T_1) = 1$, for which the first-stage term $(n_1 + \frac{1}{P_{\text{select}}(n_1,T_1)} - 1)$ simplifies to n_1. One may consider adding a third stage in an attempt to replace a longer second-stage code n_2 with two shorter codes n_2' and n_3. However, such a modification is only advantageous if the resulting product

$$\left(n_2' + \frac{1}{P_{\text{select}}(n_2', T_2')} - 1\right)\left(n_3 + \frac{1}{P_{\text{select}}(n_3, T_3)} - 1\right) \tag{41}$$

is smaller than the original second-stage term $(n_2 + \frac{1}{P_{\text{select}}(n_2,T_2)} - 1)$. Since the shortest allowable code has length 7, we evaluate the best-case scenario for the third-stage approach by assuming both n_2' and n_3 are 7. This yields a minimal product value $7 \times 7 = 49$ of term (41). Our analysis indicates that, for all practical values studied (e.g., $n_2 \leq 15$), the original second-stage term remains well below this threshold. Therefore, adding a third stage does not reduce the total SRAM cost and is suboptimal for the typical error rates considered. Furthermore, introducing a third stage increases the total number of decoding operations by $n_k n_1 n_2'$. This additional overhead is only justifiable in extreme cases, such as very high bit-flip probabilities p_e, where much longer codes (e.g., $n_2 \gtrsim 47$) become necessary.

[2] In this analysis we do not distinguish between the ordering of shorter or longer codes across stages, as the combinations are symmetric in the total SRAM size, as confirmed by Fig. 2.