A Reliable PUF in a Dual Function SRAM

Mohd Syafiq Mispan[a,b], Shengyu Duan[a], Basel Halak[a], Mark Zwolinski[a,*]

[a] *School of Electronics and Computer Science, University of Southampton, Highfield, Southampton, SO17 1BJ United Kingdom*
[b] *Universiti Teknikal Malaysia Melaka, Fakulti Teknologi Kejuruteraan Elektrik & Elektronik, Centre for Telecommunication Research & Innovation (CeTRI)*

Abstract

The Internet of Things (IoTs) employs resource-constrained sensor nodes for sensing and processing data that require robust, lightweight cryptographic primitives. The SRAM Physical Unclonable Function (SRAM-PUF) is a potential candidate for secure key generation. An SRAM-PUF is able to generate random and unique cryptographic keys based on start-up values by exploiting intrinsic manufacturing process variations. The reuse of the available on-chip SRAM memory in a system as a PUF might achieve useful cost efficiency. However, as CMOS technology scales down, aging-induced Negative Bias Temperature Instability (NBTI) becomes more pronounced resulting in asymmetric degradation of memory bit cells after prolonged storage of the same bit values. This causes unreliable start-up values for an SRAM-PUF. In this paper, the on-chip memory in the ARM architecture has been used as a case study to investigate reliability in an SRAM-PUF. We show that the bit probability in a 32-bit ARM instruction cache has a predictable pattern and hence predictable aging. Therefore, we propose using an instruction cache as a PUF to save silicon area. Furthermore, we propose a bit selection technique to mitigate the NBTI effect. We show that this technique can reduce the predicted bit error in an SRAM-PUF from 14.18% to 5.58% over 5 years. Consequently, as the bit error reduces, the area overhead of the error-correction circuitry is about 6× smaller compared to that without

*Corresponding author
Email address: mz@ecs.soton.ac.uk (Mark Zwolinski)

a bit selection technique.

Keywords: Aging, physical unclonable function, SRAM, reliability.

1. Introduction

The emerging Internet of Things (IoTs) has raised major security concerns as it enables communication and data transfer through a network of interconnected devices such as smartphones, smart-meters and smart-homes that have access to personal data. The mobility and interconnectivity of IoT devices make them vulnerable to theft and tampering. Therefore, providing fundamental security services such as authentication and encryption for IoT devices is crucial. IoT devices are resource-constrained, thus providing security introduces a significant challenge.

In response to this challenge, a class of cryptographic primitives called Physical Unclonable Functions (PUFs) has been proposed as a promising technology for lightweight embedded security [1, 2, 3, 4]. PUFs map a set of challenges to a set of responses, also known as Challenge Response Pairs (CRPs) [5]. A PUF provides uniqueness, unclonability and randomness for generating cryptographic keys by exploiting inherent manufacturing process variations. The variations in devices and interconnects are stochastic in nature and therefore, nominally identical PUFs from the same manufacturing process provide unique and apparently random sets of responses.

Cryptographic keys, generated from PUFs, have been shown to be useful in providing robust and lightweight security for IoT devices [6, 7, 8, 9]. Though a wide range of PUF techniques exists, the most investigated solutions have been based on the use of random start-up values (SUVs) of SRAM-PUFs to generate random, unique and unpredictable cryptographic keys [1, 2, 6, 8]. The random SUVs are generated in intermittently powered devices when they are powered-up before use [2]. Nonetheless, dedicated memory for security introduces an area overhead and hence makes the cost unacceptable and could hinder the widespread adoption of PUFs.

Several pieces of work [6, 10, 11, 12] have suggested reusing the existing on-chip SRAM as a PUF to reduce the area overhead and achieve cost efficiency for resource-constrained devices. Schaller et al [10] and Kohnhäuser et al [11] proposed reusing the available on-chip SRAM in an ARM-based, low-end microcontroller as a PUF for firmware protection. This requires a modification to the boot-loader to extract and decrypt the keys when the device is powered-up. A similar idea to protect the firmware authenticity and integrity has been proposed, in which the L1-cache is reused as a PUF [6]. Elsewhere, Bacha et al [12] reused the L2-cache in an Intel Itanium II processor as a PUF for an authentication mechanism. The CRPs which are used for server-client authentication process are generated by exploiting the error which occurs in a cache line when the nominal supply voltage of the L2-cache is reduced to a few mV.

All of the above studies show that it is practical to reuse the existing on-chip SRAM in a system as a PUF. However, the SRAM is also used to store an application's instruction and data. When the SRAM is used as a memory, the prolonged storage of the same bit patterns can cause asymmetric degradation of bit cells because of Negative Bias Temperature Instability (NBTI). For SRAM-PUFs, this means that the start-up values (SUVs) may become unreliable [2, 13]. In order to use an SRAM-PUF for cryptographic key generation, it must be able to generate error-free keys and this can be achieved by using error-correction codes (ECCs) [1, 14]. However, the area of the ECC increases as the errors in the SUVs increase, which can be seen as a disadvantage for an SRAM-PUF that is used in a resource-constrained IoT devices. Preprocessing the PUF's response is one of the techniques to reduce the area of the ECC [15, 16]. Such a technique has been exploited by Maiti et al [17], in which one pair of ring oscillators (RO) is selected from a group of ROs such that they have the maximum difference in frequency, in order to increase the robustness of the RO-PUF against aging.

Motivated by the reliability issue of SRAM in dual use as a memory and PUF and the cost of preprocessing techniques, the focus of this paper is to propose bit selection as a preprocessing technique to increase the reliability of the PUF in dual function SRAM. To achieve that, we have analyzed the effect

of asymmetric stress on the SUVs in both the instruction and data caches. We used a 32-bit ARMv8 architecture as a case study. We show that we can predict the signal probability pattern in a 32-bit ARM instruction cache (i-cache) and hence determine the appropriate bits to use for reliable SUVs. Therefore, we suggest using i-cache as both memory and a PUF to save silicon area, and use bit selection to select only reliable bits.

We present three major contributions in this paper.

1. In Section 3.1, we present a detailed analysis of the stress distribution in the instruction and data caches. The analysis of the instruction cache shows that the distribution patterns for a given processor are predictable. The data cache shows much less predictability.
2. We also show in Section 3.1 that the effect of NBTI on the SUVs of the bits of an SRAM-PUFs is not uniform, but that it is predictable in a 32-bit ARM instruction cache.
3. In Section 3.2, we propose a bit selection technique to select only bit cells that have close to a 50% probability of storing a value of '1'. We show that this technique can significantly reduce the NBTI effect on SRAM-PUFs.

2. Background and Related Work

As noted in Section 1, the SUVs of an SRAM-PUF after powering up can be used to generate random and unique cryptographic keys. To save area, SRAM can be used as both a memory and a PUF. However, the reliability of SUVs degrades due to the impact of NBTI when SRAM is used as a memory. In this section, we explain in detail the procedure and the cost of generating error-free cryptographic keys from random SUVs of an SRAM-PUF, the impact of NBTI on SRAM, as well as work on aging in SRAM-PUFs.

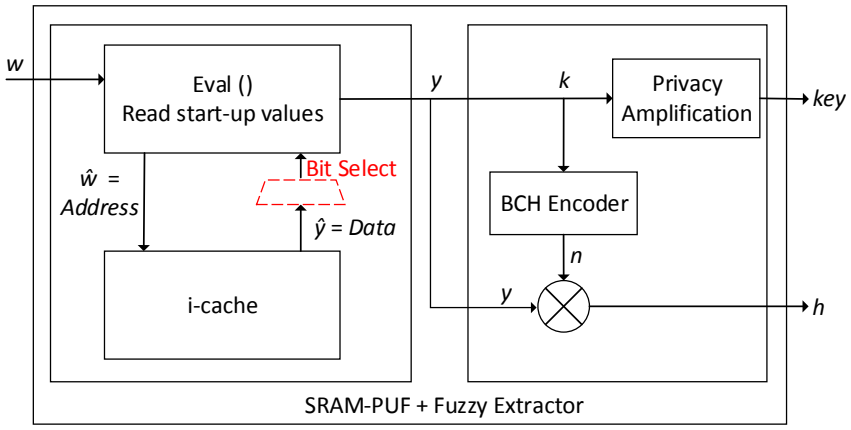
2.1. Key Generation

Key generation from random SUVs in caches has been described by Hoffman et al [6] and Schaller et al [10], where, respectively, the L1 cache and the Level-3

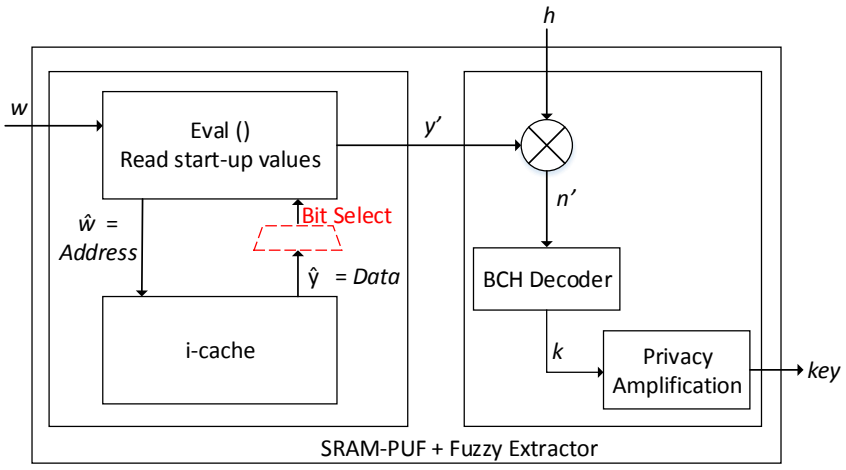
on-chip SRAM (L3 OCM SRAM) have been used. In both studies [6, 10], the key is generated only once and regenerated at each device turn-on time. To generate error-free keys from SRAM SUVs, fuzzy extractors are used, which include ECCs to handle uncertainty in the SUVs, and privacy amplification to achieve maximum entropy from hash functions [1, 18], as shown in Figure 1 (the bit select block will be discussed in detail in Section 4.1). A BCH scheme is used as an ECC to generate error-free cryptographic keys [1]. The procedure consist of two phases: (a) *Enrollment*, and (b) *Reconstruction* [6, 18]. The *Enrollment* phase occurs just once, in a controlled and trusted environment before the device is delivered to the market. During this phase, random addresses, w, of the i-cache are used to select the SUVs, y. A BCH encoder receives k which is a subset of the SUVs, y, and generates a codeword n. The *key* is produced by $hash(k)$ and the helper data, h, is computed as $n \oplus y$. At the end of this phase, the helper data, h, and the set of addresses, w, are stored in the memory. In the *Reconstruction* phase, which occurs every time the system is powered-on [6, 10], the same PUF is measured again (i.e., the noisy response y') and a noisy codeword n' is computed as $y' \oplus h$. A BCH decoder is used to correct n' and then the *key* is recovered after performing $hash(k)$.

2.2. Area Estimation of Error Correction Code

As shown in Figure 1, the BCH scheme is used as an ECC to generate error-free keys. The BCH scheme is given as $[n, k, t]$ where n raw bits (codeword) are required for a k bit message and the scheme can correct up to t bits in error. For example, using a codeword $n = 127$, a BCH scheme can fix up to $t = 31$ error bits for a message $k = 8$. For $t > 31$, one has to choose the next available codeword that has the capability to fix more errors. The gate equivalent (GE) area of the BCH scheme has been calculated [19] for $5 \leq m \leq 7$ and $1 \leq t \leq 3$, where codeword $n = 2^m - 1$. It has been suggested elsewhere [20] that the area increases linearly with the number of errors, but this has to be assumed to be the best case. Using this data, the lin-log plot of the area (y-axis) versus the codeword n (x-axis) can be derived and extrapolated to estimate the area for



(a) Enrollment phase



(b) Reconstruction phase

Figure 1: Procedure for cryptographic key generation [18].

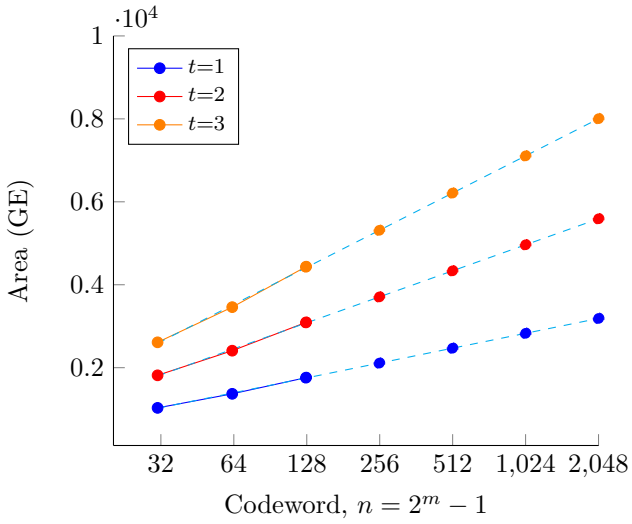


Figure 2: Area (GE) of the BCH scheme

.

$8 \leq m \leq 11$. Figure 2 shows the calculated area (solid lines) from [19] and the estimated area (dashed lines) by extrapolation. For the BCH scheme, each codeword has a maximum capability to fix a certain number of errors. Figure 2 only shows the area for $1 \leq t \leq 3$. To estimate the area for $t > 3$, the relationship between the area and the number of errors would have to be found.

2.3. Impact of NBTI on SRAM

NBTI manifests itself as an increase in the threshold voltage (V_{th}) when a PMOS device is negatively biased [21]. In an SRAM cell, only the pull-up transistors, MP1 and MP2 (Figure 3), would suffer from NBTI [22]. Since MP1 and MP2 are each one side of the cross-coupled inverters, only one can be under NBTI stress at any time. This might result in unbalanced V_{th} degradation of the two transistors and thereby lead to a mismatch. A metric, the PUF Static Noise Margin Ratio ($PSNM_{ratio}$), is used to model the SRAM SUV in SPICE [23]. The $PSNM_{ratio}$ is the ratio of the two "eyes" (NM_1/NM_0) in the SRAM static noise margin (SNM) plot, Figure 4. Cells with a $PSNM_{ratio}$ significantly

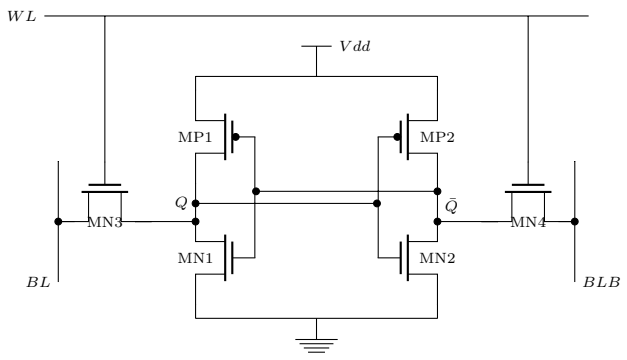


Figure 3: 6-T SRAM cell circuit.

higher or lower than 1 are more likely to start up with '1' or '0', respectively, and therefore are more reliable in that the SUV is more consistent.

Because of NBTI, the longer a cell stores a state, the more probable it is that its SUV will become the opposite of the stored value [24, 25, 26]. For instance, transistor MP1 in Figure 3 would be turned on and stressed more and thereby become weaker if the cell always stores a '1'. If the cell is powered-up after having stored a '1' over a period of time, node Q is less likely to be pulled up to '1' than it was before. We used the MOSFET model reliability analysis (MOSRA) model built into HSPICE [27] with a TSMC 65-nm CMOS technology to model this. Figure 5 shows the $PSNM_{ratio}$ shift due to NBTI on one SRAM PUF cell. The absolute values may vary because of modeling and environmental variations but the trend still holds. The curves indicate that a cell tends to be less reliable as the $PSNM_{ratio}$ tends to 1 or if the cell is more likely to store the same value as its SUV. This observation is supported by experimental data [2, 28, 13]

2.4. Aging in SRAM-PUFs

The effects of NBTI on the reliability in a 65-nm technology have been extensively studied [29, 24]. These analyses show that an SRAM-PUF would suffer up to an 8% bit error rate after 4.5 years. Elsewhere, Selimis et al [30]

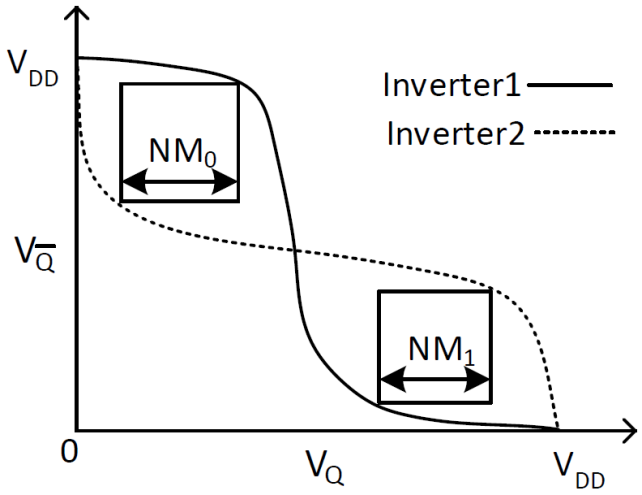


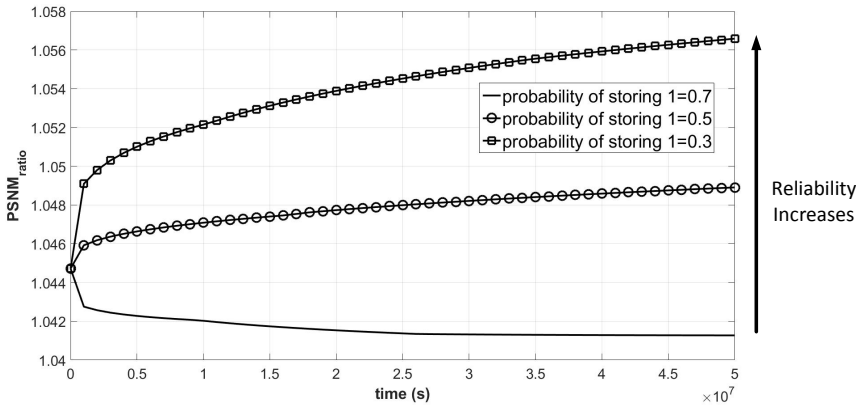
Figure 4: Inverter characteristics for SRAM cell and SRAM static noise margins.

reported that NBTI causes up to a 14% bit error rate on SRAM-PUF after 4.7 years for a 90-nm technology.

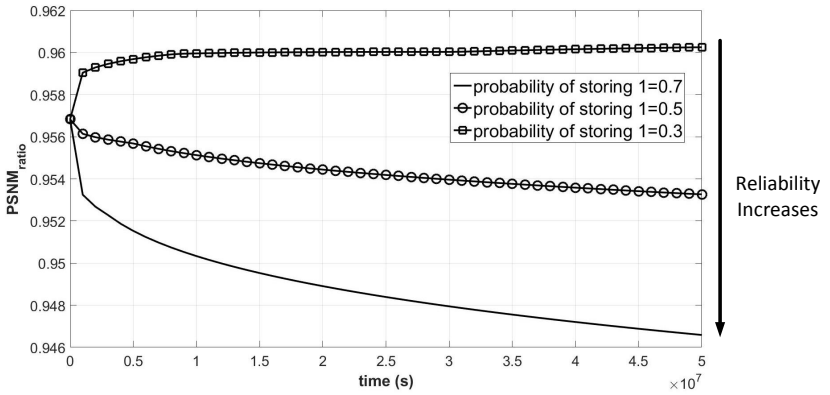
One study analyzed 16 possible (anti-)aging configurations [24]. Of these, reinforcing the power-up state using the long-term aging data (the inverse of the corrected enrollment state using ECC) was found to be the best solution. However, to use the inverse of the corrected enrollment state as an anti-aging strategy is not really practical if the SRAM is not solely dedicated to the PUF function.

The spatial correlation of all the bit cells under environmental variations (temperature and voltage) has also been studied [31] and it was proposed to select only those fully-biased bit cells, eliminating the partial-skewed and neutral-skewed cells. The proposed bit select algorithm increases the robustness of an SRAM-PUF against environmental variations, but it is still susceptible to reliability issues due to NBTI, when used both as memory and as a PUF. It is notable that all of the above studies assume the use of SRAM for the PUF function only and not for regular memory use.

On the other hand, a number of studies have proposed NBTI mitigation for SRAM specifically used as memory components like register files and caches. A



(a) SRAM cell with an initial SUV of 1



(b) SRAM cell with an initial SUV of 0

Figure 5: $PSNM_{ratio}$ shifts due to NBTI for SRAM cells (T=300K, V_{dd}=1.2V, V_{tp}=-0.436V, $\Delta V_{tp0} = 50mV$).

periodic cell flipping scheme has been used to create a more symmetric signal probability in an SRAM cell to reduce the impact of NBTI [32, 33, 34]. By definition, these techniques could also help to mitigate NBTI-induced SUV shifts when an SRAM is used in a dual function mode. Nevertheless, state-of-the-art techniques do not guarantee a fully balanced probability (i.e., 50%) for a cell, although the probability may become closer to 50%. In such cases, some SRAM cells could still store the same values for a long time, affecting their SUVs when used as PUFs. This situation will be demonstrated in the following section.

3. Preprocessing Approaches

Reusing the on-chip SRAM in a system as a PUF, as suggested in [6, 10, 11, 12], can be cost efficient, especially for resource-constrained devices. However, dual use of SRAM as a memory and a PUF is not straightforward because NBTI results in asymmetric degradation of memory bit cells and reduces the SUV reliability over time. In our work, the on-chip memory and, in particular, the instruction cache in an ARMv8 architecture, has been used as a case study. The NBTI analysis based on the signal probability pattern of the instruction cache suggests that some of the bits experience nearly symmetric stress. Symmetric stress indicates that the cross-coupled inverters age at the same rate and the intrinsic mismatch (i.e., V_{th} variations) due to the random process variations is retained. Therefore, the SUVs of these bit cells which depend on the V_{th} mismatch during the power-up process are more reliable with regard to aging. On the other hand, asymmetric stress due to aging causes one of the cross-coupled inverters to experience V_{th} degradation more than the other. As a consequence, the SUVs of the bit cells that experience asymmetric stress are biased either to '0' or '1' which may result in unreliable SUVs. Hence, to increase the SUV reliability for PUF usage, bit selection is used as a preprocessing technique by only selecting bit cells that experience symmetric stress. The bit selection technique is most suitable for a system that runs a specific application, such as an embedded system. Figure 6 shows an overview of this technique, which is divided into

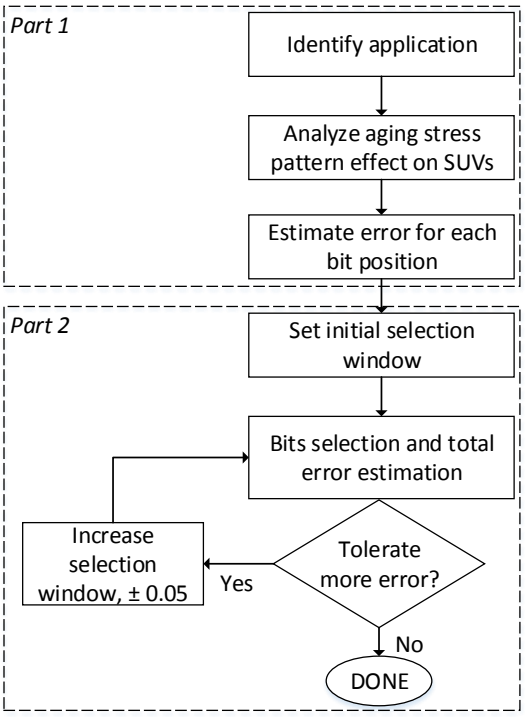


Figure 6: Overview of a bit selection technique

two parts. The two parts will be discussed in Sections 3.1 and 3.2, respectively.

3.1. Stress Distribution in SRAM Caches

3.1.1. Stress Patterns

Previous studies of instruction set usage indicated that only a small subset of instructions is executed by most applications [35, 36]. We expect that the overall bit probabilities in the instruction cache (i-cache) will be distributed unevenly. We ran a set of simulations of the i-caches for an ARMv8 architecture using gem5 [37]. Table 1 gives the cache configuration. 16 benchmark programs were chosen from the MiBench [38] and llvm [39] benchmark suites, covering a number of different applications such as automotive and industrial, network, security, and

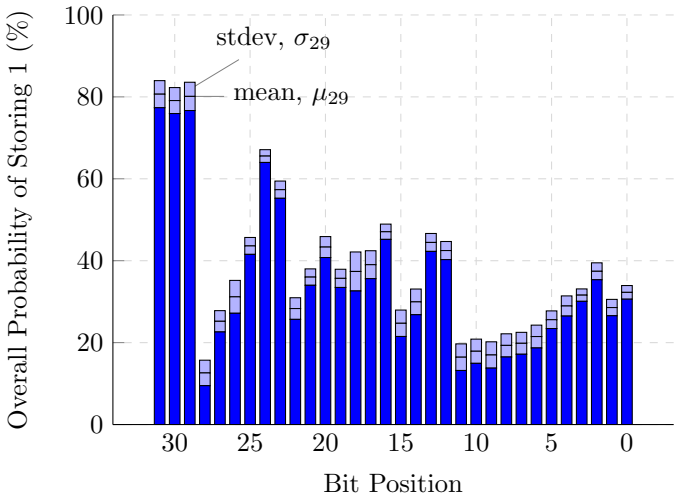
telecommunications. Therefore, the relation between the bit storage pattern of an i-cache and the running programs could be investigated.

For each benchmark program, each cache access was dynamically traced during program execution, such that the bit probabilities for each cache line were updated every time that it was replaced. The probability distribution was produced by averaging the final bit probabilities over the whole cache. Once the probability distributions for all 16 benchmark programs were found, the overall probabilities of storing '1' are computed by averaging the 16 probability distributions. The results are shown in Figure 7 for two extreme cases of cache policy – direct mapped cache and fully associative, which give the least and the most balanced block usage, respectively. In each case, some bits preserve the same values in certain locations. As can be seen, the patterns are generally independent of the programs and of associativity, and therefore are predictable.

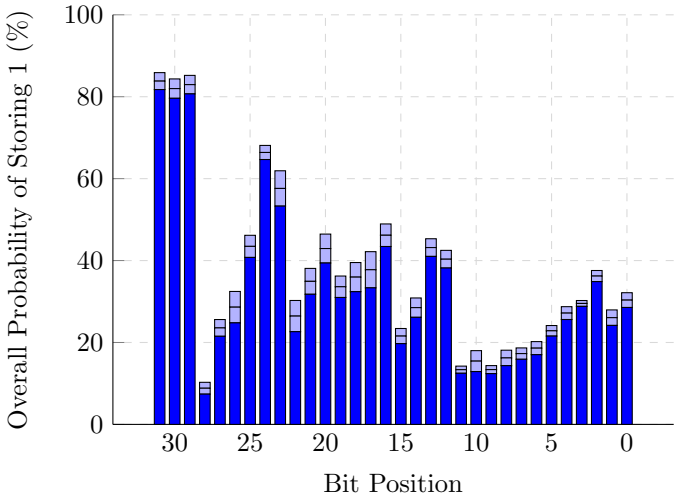
Table 1: Cache configuration.

Cache size	8kB
Block size	16-Byte
Number of block	512
Addressing scheme	Word (4-byte) addressing
Computer architecture	32-bit ARM

For the data cache (d-cache), it might be expected that the overall probability at each location is around 50% due to the randomness of data. Therefore d-caches would be more suitable as PUFs, compared with i-caches. Using gem5, we observed the overall bit probabilities in the d-cache by tracing each cache access. Figure 8 presents the probability patterns in a direct mapped d-cache for four benchmarks. Although the distributions are more balanced than in the i-cache, the probabilities of storing a '1' are less than 50% for all these benchmarks, which indicates asymmetric stress in the d-cache. Moreover, unlike in the i-cache, the distribution is highly application-dependent. It is possible that other applications may produce different distributions (with bit probabilities



(a) Direct mapped cache



(b) Fully associative cache
(replacement policy:most recently used)

Figure 7: Mean and standard deviation values for the probability of storing a '1' in i-cache over 16 benchmarks.

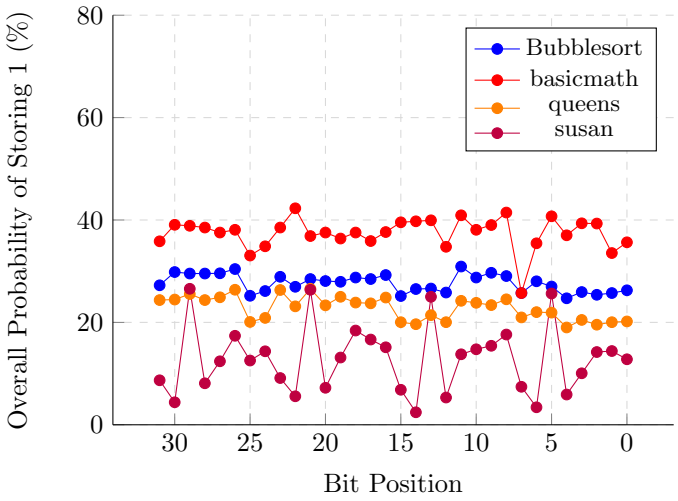


Figure 8: Mean values for probability of storing '1' in d-cache running 4 benchmarks.

equal to or higher than 50%), but we conclude that we cannot easily predict the reliability of a d-cache nor its performance as a PUF. This stress pattern analysis indicates that an i-cache is more suitable for use as a PUF and so is the focus of this study.

3.1.2. Cell Flipping

As was mentioned in Section 2.4, a cell flipping mechanism is often used in SRAM memories to reduce the impact of NBTI. Figure 9 gives an SRAM array with a typical cell flipping interface which uses a "flip on access" mechanism, whereby the write and read paths are controlled by a counter and an inversion flag register [32, 33, 34]. The counter is used to periodically invert the signal flip_{in} to flip the input data. The inversion flag indicates if the data in a cell is true or complementary, and therefore ensures any data read out is correct.

We used the cell flipping scheme shown in Figure 9 in the i-cache of an ARM architecture to measure the stress patterns. The stress patterns of the direct mapped caches with and without the cell flipping scheme are shown in Figure 10, when simulated with the fft benchmark. As can be seen, the probabilities

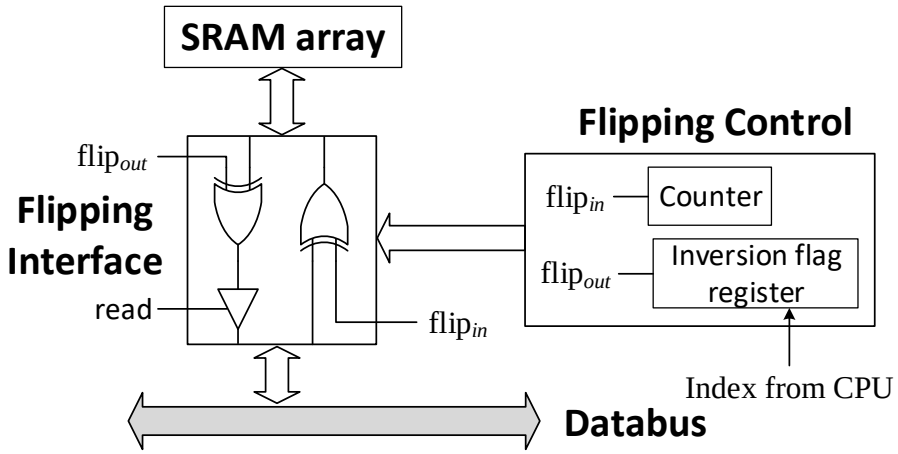


Figure 9: A typical cell flipping SRAM array based on "flip on access" (FOA) mechanism [32, 33, 34].

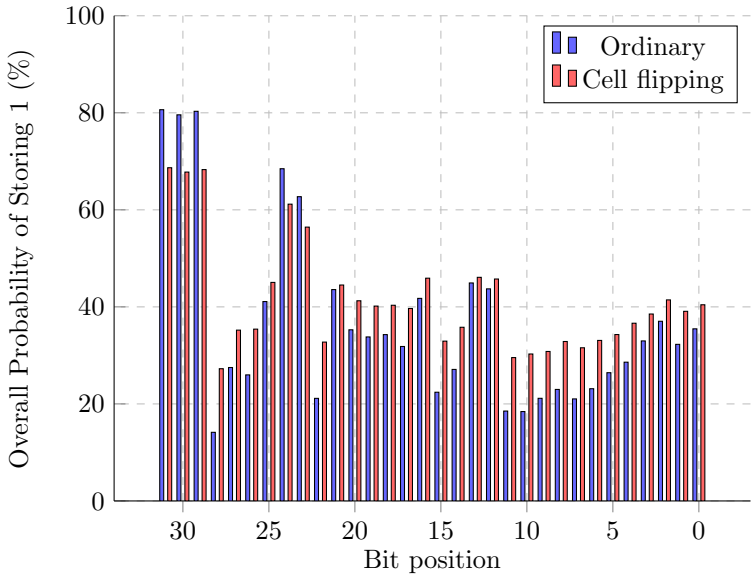


Figure 10: Probability of storing a 1 in direct mapped i-caches with/without cell flipping scheme.

of all positions become more balanced, but some like bits 28 to 31 are still far away from 50%. This is because the flipping control circuit only manipulates the cache access paths (i.e., flip on access), and thus the cell flipping mechanism is only activated during the cache access. A cache memory is designed to store the most useful instructions or data as long as possible. In such a case, most cache blocks are not frequently replaced and therefore the stored data may be seldom flipped. Those cells that are infrequently replaced may still preserve the same values for a long time, causing asymmetric probabilities and severe SUV shifts.

3.1.3. Start-up Values

Without the effect of NBTI, the randomness of SUVs in SRAM cells is determined by the process-dependent variations between the cross-coupled inverters. To simulate the process variations, an SRAM i-cache has been modeled in a low-k 65-nm technology. To get a good distribution of process variations, a Monte Carlo simulation was performed for 100 SRAM instances. However, to run a SPICE simulation of the complete circuit of an 8kB SRAM, as shown in Table 1, is prohibitive, and therefore, only 32 bits x 64 rows were simulated and analyzed. The simulation was performed with the BSIM4 (V4.5) transistor model at a nominal supply voltage 1.2V and a temperature 25°C, using the 3σ variation in the technology design kit. The ramp-up time of the supply voltage is fixed at $1ms$. When the supply voltage is ramped up from 0V to 1.2V the SUVs of SRAM cells settle randomly at either '0' or '1' due to the process variations. To determine the aging impact on SUVs due to NBTI, a simulation procedure as described in [5] was used. The duty cycle for the 32-bit SRAM cells was defined according to the signal probabilities in Figure 7. Then, the V_{th} degradation for every bit cell was simulated, using HSPICE MOSRA [27], for 5 years with 100% activity factor.

Based on the Monte Carlo simulation of 100 SRAM instances, the randomness of SUVs in SRAM cells is shown in Figure 11. SRAM cells have an even distribution of '1' and '0' for all bit cells (see Figure 11 (a)), therefore, they

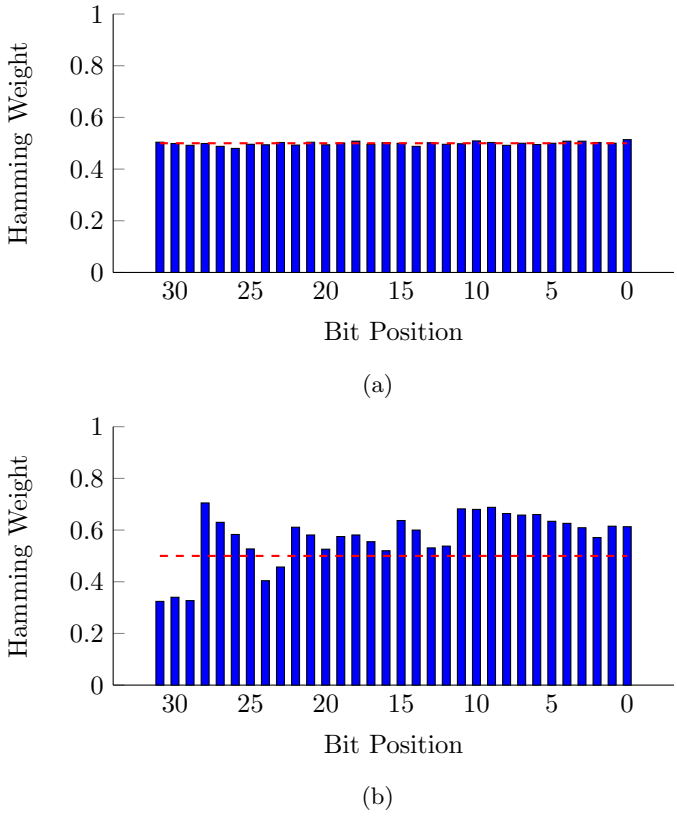


Figure 11: Distribution of '1' and '0' (a) fresh (b) 5 years aging based on the mean probability of storing '1'.

are random. In other words, the Hamming Weight (HW) for each bit is approximately 0.5. However, when they undergo NBTI aging with the signal probabilities given in Figure 7 (a), the distribution of '1's and '0's become uneven: bits 29, 30 and 31 have a high probability of storing '1's. As a result of the prolonged storage of same bit value, after 5 years they are likely to power-up to '0', as illustrated in Figure 11 (b). Conversely, bits 9 to 11 and bit 28 are likely to power-up to '1'. The bits that have close to a 50% probability of storing '1', such as bits 12, 13, 16, 20, 23 and 25, still show an even distribution of '1's and '0's.

Based on the distribution of SUVs at fabrication time and after 5 years, the bit error for every bit cell is calculated using the intra-chip Hamming Distance NBTI (Intra-HD_{NBTI}), as shown in Figure 12. The Intra-HD_{NBTI} is derived from the Intra-HD. The Intra-HD is a measure of the reproducibility of a PUF response given the same challenge (the SUV in the case of an SRAM-PUF), under aging, temperature or voltage variations. For the challenge C, a single chip, represented as i, has an n-bit reference response $R_i(n)$ at room temperature and a nominal supply voltage (i.e., the reference condition). The same challenge C is applied to the chip i at different condition to obtain the n-bit response, $R'_{i,j}(n)$. Hence, the average Intra-HD for m samples is defined as (1) [40].

$$\text{Intra–HD} = \frac{1}{m}\sum_{j=1}^{m}\frac{\text{HD}(R_i(n), R'_{i,j}(n))}{n} \times 100\% \tag{1}$$

The error due to aging is called the Intra-HD_{NBTI}, henceforth, to distinguish it from the Intra-HD due to temperature or voltage variations (as discussed in Section 3.2.1 below) and is given as (2).

$$\text{Intra–HD}_{\text{NBTI}} = \frac{1}{m}\sum_{i=1}^{m}\frac{\text{HD}(R_{i.fresh}(n), R_{i.aged}(n))}{n} \times 100\% \tag{2}$$

As expected, because of the symmetric stress experienced by bits 12, 13, 16, 20, 23 and 25, the bit error count in those positions is much less than for the other bits. One might argue that NBTI can be used to increase the reliability

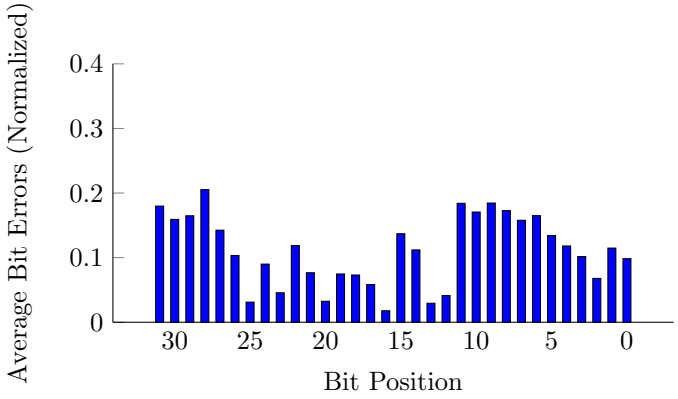


Figure 12: Average bit errors based on the mean probability of storing '1'.

of SRAM-PUFs by storing the inverse value of SUVs as in [13, 24]. However, that only works if the SRAM is dedicated for use as a PUF only. Here, we use the SRAM as both an instruction cache and a PUF. Because of the random SUVs, choosing only the bit cells that experience balanced stress can retain the intrinsic mismatch of the inverters in the cell. Nevertheless, these bit cells are exposed to environmental variations such as temperature or supply voltage variations, as will be discussed in Section 3.2.1. At this point, we have discussed the first part of the preprocessing procedures (Figure 6). The second part of the preprocessing procedure will be discussed next.

3.2. Bit Selections

As discussed in Section 2, in order to extract a key from a PUF we need a certain number of bits. Using all of an SRAM gives us the maximum number of bits to choose from but as we have seen from Section 3.1.3, the SUVs of many bits become unreliable over time. Therefore, the purpose of bit selection is to find as many reliable bits as possible within an SRAM.

There are two important observations from the previous analysis: (i) the reliability of the SUV of a cell depends on the evenness of the stress in the cross-coupled inverters; (ii) 6 of the bit cells in this example have the fewest bit

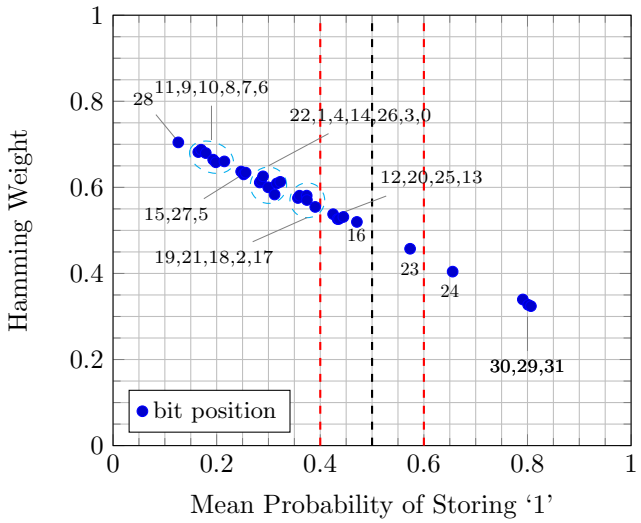


Figure 13: The relationship between the HW and the mean probability of storing '1' for all 32 bits.

errors as the system ages. These are bits 16, 13, 25, 20, 12 and 23 (ranked in ascending order of their bit error count). The idea of a bit selection technique is to select only those bits for the PUF that have close to a 50% probability of storing '1' values. Ultimately the selection of these bits reduces the overall bit error rates. To perform the second part of the preprocessing procedure, which is the bit selection, the relationship between the Hamming Weight and the mean probability of storing '1' has to be found. By taking the HW from Figure 11 (b) and the mean probability of storing '1' from Figure 7 (a), the relationship between these two parameters for all 32 bits can be seen, Figure 13. The ideal case is when a bit cell has a 50% probability of storing '1', i.e., HW=0.5. In Figure 13, as the mean probability of storing '1' reduces, the HW increases, and vice versa. In order to evaluate the increase/decrease in the bit error rate for a selection of bits as the HW reduces or increases, first, we chose the lower boundary (LB) and upper boundary (UB) at 0.4 and 0.6 (i.e., initial selection window), respectively, as highlighted by the two dashed red lines.

Note that the six bits that have the fewest bit error rates after aging are

within these two boundaries. These selected bits are represented as S1 and their corresponding bit error rates are shown in Table 2. The bit error rate in Table 2 is calculated by taking the sums of the average individual bit errors in Figure 12 and dividing by the total number of bit cells. As stated in Section 3.1.3, this is the Intra-HD_{NBTI}. Next, the margin of the selection window is widened by ± 0.05. The aim is to maximize the number of bits (because that reduces the extraction time to generate the whole e.g., 128-bit key) while keeping the error below the maximum error tolerance value. As can be seen from Table 2, as the margin of bit selection increases, the total bit error rate increases. The selection set S7 represents all 32 bits (i.e., without a bit selection technique) which has the highest bit error rate of 11.13%.

Table 2: Bit Error Rate (%) of Bit Selection Combination Based on Mean Probability of Storing '1'.

Selection Set	LB	UB	Selected Bits	Bit Error Rate (Intra-HD_{NBTI}) (%)
S1	0.40	0.60	12,13,16,20,23,25	3.30
S2	0.35	0.65	S1 + 2,17,18,19,21	4.98
S3	0.30	0.70	S2 + 0,3,24,26	6.28
S4	0.25	0.75	S3 + 1,4,5,14,22,27	8.01
S5	0.20	0.80	S4 + 6,15,30	8.93
S6	0.15	0.85	S5 + 7,8,9,10,11,29,31	10.83
S7	0.10	0.90	S6 + 28	11.13

To determine which selection set is the most suitable for NBTI mitigation while still fulfilling the maximum error tolerance value, it is important also to take into consideration the spread of the probability of storing a '1'. Therefore, $\pm 3\sigma$ is used to give the worst spread of the probability of storing '1' and to select the best selection set that fulfills that criterion. The $\pm 3\sigma$ probability of storing '1' is calculated based on the mean and standard deviation probability of storing

'1' from Figure 7 (a). Further, the procedure given in Section 3.1.3 is re-applied to calculate the average bit error for every bit position at $\pm 3\sigma$ conditions. Then, the total bit error rates for selection sets S1-S7 are calculated, as listed in Table 3.

Table 3: Bit Error Rate (%) of Bit Selection Combination Based on Mean and $\pm 3\sigma$ Probability of Storing '1'.

Selection Set	Intra-HD_{NBTI} (%)		
	Mean	-3σ	+3σ
S1	3.30	5.58	2.02
S2	4.98	8.39	2.23
S3	6.28	9.39	3.58
S4	8.01	11.43	4.88
S5	8.93	12.14	5.98
S6	10.83	13.77	8.12
S7	11.13	14.18	8.35

From Table 3, the overall bit error rate is worst at -3σ. In effect, all the bits would be shifted to the left in Figure 13, away from the 50% probability of storing '1'. At $+3\sigma$, all the bits are shifted to the right, closer to the 50% probability of storing '1', making the overall bit error rate better than the mean condition. Notice from Table 3 that the bit error rates for all 32 bits (S7) are within the range of 8.35% to 14.58% after 5 years. Our aging results are quite close to the experimental analysis reported elsewhere [29, 24], where the analyses show that an SRAM-PUF would suffer up to an 8% bit error due to NBTI after 4.5 years with the same 65-nm technology. Elsewhere, NBTI causes up to a 14% bit error on SRAM-PUF after 4.7 years in a 90-nm technology [30]. We do not neglect the fact that the differences might arise from the data dependency of NBTI stress and the aging model differences. Nevertheless, the effect of NBTI on SRAM cells and the overall trend still hold.

With a bit selection technique, the bit error rate varies depending on the

selection window. As mentioned earlier, the aim is to maximize the number of selected bits while keeping the error well below the maximum error tolerance value. For a proof of concept, in our study we set the maximum error tolerance due to NBTI at 6%. Therefore, we have chosen selection set S1 since the bit error rate is $< 6\%$ considering the probability of storing '1' for both the mean and $\pm 3\sigma$ conditions. As discussed in Section 2.2, a reduction in the area of the ECC can be achieved if the error is minimized. Hence, we should expect that S1 could reduce the area overhead of the ECC since the bit error rate is much smaller than that without a bit selection technique (i.e., using all 32 bits).

The above analysis shows that by selecting only those bits that experience balanced stress when the SRAM is being used as a memory we can reduce the overall bit error when the SRAM is used as a PUF. However, there are other sources of variations that could cause errors in the PUF's response, in particular temperature and voltage fluctuations [17]. The size of the ECC has to be determined by the maximum error considering aging, temperature and voltage variations. Therefore, the effect of environmental variations on set S1 and on all 32 bits, set S7, will be discussed next.

3.2.1. Temperature and Voltage Variations

The setup described in Section 3.1.3 is used to simulate the temperature and voltage variations. In this analysis, the voltage variations refer to the variations in the power supply voltage. A measured response at a nominal supply voltage 1.2V and a temperature 25°C has been used as a reference. Intra-HD, (1), is used to calculate the bit error due to the temperature and supply voltage fluctuations.

Table 4 shows the bit error percentage for a temperature range of -40°C to 85°C and a voltage range 1.2V$\pm$ 10%. The maximum bit error caused by the temperature and voltage fluctuations is 6.14%. The bit error value in our findings is comparable to the experimental value reported elsewhere [29, 41], where the maximum error rate is about 8% with the same temperature range and at a nominal supply voltage 1.2V for a 65-nm technology node.

Table 4: Bit Error (%) Comparison at Different Temperatures and Voltages.

Conditions	Selection Set	
	S7	S1
$1.2V, -40°C$	5.91	5.90
$1.2V, 85°C$	1.31	1.32
$1.32V, -40°C$	6.14	6.12
$1.32V, 85°C$	1.31	1.32
$1.08V, -40°C$	5.65	5.67
$1.08V, 85°C$	1.31	1.32

From our analysis, we observe that under a fixed ramp rate, variations in the final voltage to which the supply is ramped have almost no impact on the reliability of the SUVs; this is consistent with that seen elsewhere [30]. In addition, Table 4 shows that the bit error at 85°C is lower than at −40°C. A decrease in temperature increases V_{th} while also increasing the electron and hole mobilities, and vice versa [42]. During the power-up process, the temperature dependencies of V_{th} and electron/hole mobilities may counteract each other, which could lead to unstable SUVs [2].

4. Hardware Cost and Implementation

4.1. Area Overhead of Bit Select Configurations

The bit selection can be implemented using a multiplexer in which the select signals are pre-configured depending on the target error rate and bit select configuration. A 32-to-1 multiplexer is required to select 1 bit out of 32 bits from the output bus of the instruction cache. Thus, the bit select block can be constructed using a 6×32-to-1 multiplexer for selecting the S1 bits: 16, 13, 25, 20, 12 and 23. In the procedure for cryptographic key generation shown in Figure 1, the bit select block is inserted before the SUVs, $\widehat{y}$, are input into

the evaluation block, highlighted in red. 280 GE are required for a 6×32-to-1 multiplexer, as estimated using Synopsys Design Compiler.

4.2. Area Overhead of ECC

We have proposed bit error reduction techniques to overcome unreliable responses due to NBTI, but there will still be a finite error rate. As discussed in Section 2, a BCH scheme is used as an ECC to generate error-free cryptographic keys. For a $[n, k, t]$-BCH, the probability of failure for a BCH scheme to correct t errors is given as [1]:

$$Target\_Failure\_Rate = 1 - \sum_{i=0}^{t} \binom{n}{i} p_b^i (1 - p_b)^{n-i} \tag{3}$$

where p_b denotes the bit error probability. From the bit error analysis of the S1 set with respect to NBTI and environmental variations (Sections 3.2 and 3.2.1), the maximum bit error of 6.12% is caused by the environmental variations and hence, will be used to determine the number of raw bits, n, required for the BCH scheme. For this calculation, we refer to Guajardo et al [1], where the BCH scheme has to generate at least 171 error-free bits before being used as the input to the hash function to produce a 128-bit cryptographic key (Figure 1). We assume a target failure rate of $\leq 10^{-6}$. Using (3), 1524 raw bits are needed for the S1 selection set using a [127,15,27]-BCH. Selecting 6 out of 32 bits for every cache line, the total number of reliable bits that can be extracted from an 8kB SRAM i-cache is 12288. Thus, there are enough bits for an ECC to generate a 128-bit key. The suggested minimum size of i-cache to be used with the S1 selection is 1kB.

Not using a bit selection technique would require more area for the BCH scheme because of an increase in the bit error rates. Considering the bit error rates due to aging and environmental variations (see Tables 3 and 4), using all 32 bits would yield a maximum error of 14.18%. This requires 4599 raw bits using the BCH scheme of [511,19,119] for a target failure rate of $\leq 10^{-6}$ and a minimum size of i-cache of 1kB. The minimum area for both BCH schemes above

is estimated based on the extrapolated data of Figure 2 and it is listed in Table 5, together with the area overhead of a bit selection technique. As can be seen in Table 5, by using a bit selection technique, the total area overhead is about 6× smaller compared to that without a bit selection technique. Notice too that the area overhead of the multiplexer to perform a bit selection technique is almost negligible. This comparison validates the claim in Section 1, that preprocessing a PUF's response, here using a bit selection technique, helps to reduce the area overhead of the ECC. Our findings in Table 5 cannot be compared with other techniques as there is no previously reported data on preprocessing techniques to mitigate the aging impact on SRAM-PUF.

Table 5: Area Comparison

Technique	BCH (GE)	Bit Selection (GE)	Total (GE)
Without Bit selection	226224	NA	226224
Bit selection	37050	280	37330

5. Conclusion

Resourced-constrained IoT devices require stringent and lightweight cryptographic primitives to secure sensitive data. SRAM-PUF is seen as a potential candidate for hardware-based key generation. Recent literature suggests reusing the on-chip SRAM in a system as a PUF to achieve a better cost efficiency. However, dual use of SRAM as a memory and a PUF turns out to be less straightforward than expected due to aging-induced NBTI. When SRAM is used as a memory, NBTI causes asymmetric V_{th} degradation, which impacts on the reliability of the SUVs. From our analysis, NBTI stress is unevenly distributed in a 32-bit ARM i-cache, but there are similar predictable patterns for different applications which could be exploited to generate reliable SUVs. In this paper, we propose a bit selection technique to mitigate the NBTI effect. We select bits

that have close to a 50% probability of storing '1' values. Thus, both sides of the SRAM cell age at the same rate and so keep the intrinsic mismatch. Our experiments show that in an ARM architecture, the bit error rate is reduced from 14.18% to 5.58% over 5 years. By using a bit selection technique, the bit error rate reduces and the area overhead of the ECC is 6× smaller compared to that without a bit selection technique, with a minimum size requirement of a 1kB i-cache. Depending on the bit select configuration and the target error rate, the selection of the bits can be pre-configured using a multiplexer. The area overhead to implement the bit selection is negligible compared with the reduction in the area overhead of the ECC.

Acknowledgments

This work was partly supported by Cisco Research Center Grant # 593688. All data supporting this study are openly available from the University of Southampton repository at http://dx.doi.org/10.5258/SOTON/397022.

[1] J. Guajardo, S. S. Kumar, G.-J. Schrijen, P. Tuyls, FPGA intrinsic PUFs and their use for IP protection, in: International Conference on Cryptographic Hardware and Embedded Systems, 2007, pp. 63–80.

[2] D. E. Holcomb, W. P. Burleson, K. Fu, Power-Up SRAM state as an identifying fingerprint and source of true random numbers, IEEE Transactions on Computers 58 (9) (2009) 1198–1210.

[3] M. S. Mispan, B. Halak, Z. Chen, M. Zwolinski, TCO-PUF: A subthreshold physical unclonable function, in: IEEE PRIME, 2015, pp. 105–108.

[4] B. Halak, Y. Hu, M. S. Mispan, Area efficient configurable physical unclonable functions for FPGAs identification, in: IEEE International Symposium on Circuits and Systems, 2015, pp. 946–949.

[5] M. S. Mispan, B. Halak, M. Zwolinski, NBTI aging evaluation of PUF-based differential architectures, in: IEEE International Symposium on On-Line Testing and Robust System Design, 2016, pp. 103–108.

[6] C. Hoffman, M. Cortes, D. F. Aranha, G. Araujo, Computer security by hardware-intrinsic authentication, in: International Conference on Hardware/Software Codesign and System Synthesis, 2015, pp. 143–152.

[7] M. Hoeberl, I. Haider, B. Rinner, Towards a secure key generation and storage framework on resource-constrained sensor nodes, in: International Conference on Embedded Wireless Systems and Networks, 2016, pp. 313–318.

[8] Intrinsic ID, SRAM PUF: The Secure Silicon Fingerprint (June 2016).

[9] B. Halak, M. Zwolinski, M. S. Mispan, Overview of PUF-based hardware security solutions for the Internet of Things, in: IEEE Midwest Symposium on Circuits and Systems, 2016, pp. 1–4.

[10] A. Schaller, T. Arul, V. Van Der Leest, S. Katzenbeisser, Lightweight anti-counterfeiting solution for low-end commodity hardware using inherent PUFs, in: Trust and Trustworthy Computing, Springer International Publishing, 2014, pp. 83–100.

[11] F. Kohnhäuser, A. Schaller, S. Katzenbeisser, PUF-based software protection for low-end embedded devices, in: M. Conti, M. Schunter, I. Askoxylakis (Eds.), Trust and Trustworthy Computing, Springer International Publishing, 2015, pp. 3–21.

[12] A. Bacha, R. Teodorescu, Authenticache: Harnessing cache ECC for system authentication, in: International Symposium on Microarchitecture, 2015, pp. 128–140.

[13] M. Bhargava, C. Cakir, K. Mai, Reliability enhancement of bi-stable PUFs in 65nm bulk CMOS, in: IEEE International Symposium on Hardware-Oriented Security and Trust, 2012, pp. 25–30.

[14] V. V. D. Leest, R. Maes, G.-J. S. Pim, P. Tuyls, Hardware intrinsic security to protect value in the mobile market, in: Information Security Solutions Europe, 2014, pp. 188–198.

[15] M.-D. M. Yu, S. Devadas, Secure and robust error correction for physical unclonable functions, IEEE Design and Test of Computers 27 (1) (2010) 48–65.

[16] C. Böhm, M. Hofer, Physical Unclonable Functions in Theory and Practice, Springer New York, 2013.

[17] A. Maiti, P. Schaumont, The impact of aging on a physical unclonable function, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 22 (9) (2014) 1854–1864.

[18] V. van der Leest, P. Tuyls, Anti-Counterfeiting with Hardware Intrinsic Security, in: Design, Automation & Test in Europe Conference & Exhibition, 2013, pp. 1137–1142.

[19] Y. Lao, B. Yuan, C. H. Kim, K. K. Parhi, Reliable PUF-based local authentication with self-correction, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems PP (99) (2016) 1–13.

[20] E. Jamro, The design of a VHDL based synthesis tool for BCH Codecs, Msc. thesis, University of Huddersfield, UK (1997).

[21] K. B. Sutaria, S. Member, J. B. Velamala, C. H. Kim, S. Member, T. Sato, Y. Cao, Aging statistics based on trapping / detrapping : Compact modeling and silicon validation, IEEE Transactions on Device and Materials Reliability 14 (2) (2014) 607–615.

[22] J. Qin, X. Li, J. B. Bernstein, SRAM stability analysis considering gate oxide SBD, NBTI and HCI, in: IEEE International Integrated Reliability Workshop, 2007, pp. 33–37.

[23] M. Cortez, A. Dargar, S. Hamdioui, G.-J. Schrijen, Modeling SRAM start-up behavior for Physical Unclonable Functions, in: IEEE Intl Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2012, pp. 1–6.

[24] R. Maes, V. van der Leest, Countering the effects of silicon aging on SRAM PUFs, in: IEEE International Symposium on Hardware-Oriented Security and Trust, 2014, pp. 148–153.

[25] A. Garg, T. T. Kim, Design of SRAM PUF with improved uniformity and reliability utilizing device aging effect, in: IEEE International Symposium on Circuits and Systems, 2014, pp. 1941–1944.

[26] C. Cakir, M. Bhargava, K. Mai, 6T SRAM and 3T DRAM data retention and remanence characterization in 65nm bulk CMOS, in: IEEE Custom Integrated Circuits Conference (CICC), 2012, pp. 1–4.

[27] Synopsys Inc., HSPICE User Guide: Basic Simulation and Analysis, California, USA (2013).

[28] A. Roelke, M. R. Stan, Attacking an SRAM-based PUF through Wearout, in: IEEE Computer Society Annual Symposium on VLSI, 2016, pp. 206–211.

[29] R. Maes, V. Rozic, I. Verbauwhede, P. Koeberl, E. V. D. Sluis, V. Van Der Leest, Experimental evaluation of physically unclonable functions in 65 nm CMOS, in: Proceedings of the European Solid-State Circuits Conference, 2012, pp. 486–489.

[30] G. Selimis, M. Konijnenburg, M. Ashouei, J. Huisken, H. De Groot, V. Van Der Leest, G. J. Schrijen, M. Van Hulst, P. Tuyls, Evaluation of 90nm 6T-SRAM as physical unclonable function for secure key generation in wireless sensor nodes, in: IEEE International Symposium on Circuits and Systems, 2011, pp. 567–570.

[31] K. Xiao, T. Rahman, D. Forte, Y. Huang, M. Su, M. M. Tehranipoor, Bit selection algorithm suitable for high-volume production of SRAM-PUF, in: IEEE International Symposium on Hardware-Oriented Security and Trust, 2014, pp. 101–106.

[32] S. Wang, T. Jin, C. Zheng, G. Duan, Low power aging-aware register file design by duty cycle balancing, in: Design, Automation & Test in Europe Conference & Exhibition, 2012, pp. 546–549.

[33] H. Amrouch, T. Ebi, J. Henkel, Stress balancing to mitigate NBTI effects in register files, in: 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2013, pp. 1–10.

[34] A. Gebregiorgis, M. Ebrahimi, S. Kiamehr, F. Oboril, S. Hamdioui, M. B. Tahoori, Aging mitigation in memory arrays using self-controlled bit-flipping technique, in: 20th Asia and South Pacific Design Automation Conference (ASP-DAC), 2015, pp. 231–236.

[35] C. Mutigwe, J. Kinyua, F. Aghdasi, Instruction set usage analysis for application-specific systems design, Intl Journal of Information Technology and Computer Science 7 (2) (2013) 99–103.

[36] A. H. Ibrahim, M. B. Abdelhalim, H. Hussein, A. Fahmy, An Analysis of x86-64 Instruction Set for Optimization of System Softwares, Intl Journal of Advanced Computer Science 5 (4) (2015) 152–162.

[37] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al., The gem5 simulator, ACM SIGARCH Computer Architecture News 39 (2) (2011) 1–7.

[38] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, R. B. Brown, Mibench: A free, commercially representative embedded benchmark suite, in: Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on, IEEE, 2001, pp. 3–14.

[39] C. Lattner, V. Adve, Llvm: A compilation framework for lifelong program analysis & transformation, in: Code Generation and Optimization, 2004. CGO 2004. International Symposium on, IEEE, 2004, pp. 75–86.

[40] A. Maiti, V. Gunreddy, P. Schaumont, A systematic method to evaluate and compare the performance of physical unclonable functions, in:

P. Athanas, D. Pnevmatikatos, N. Sklavos (Eds.), Embedded Systems Design with FPGAs, Springer New York, New York, 2013, pp. 245–267.

[41] M. Cortez, S. Hamdioui, V. Van Der Leest, R. Maes, G. J. Schrijen, Adapting voltage ramp-up time for temperature noise reduction on memory-based PUFs, in: IEEE International Symposium on Hardware-Oriented Security and Trust, 2013, pp. 35–40.

[42] P. E. Allen, D. R. Holberg, CMOS Analog Circuit Design, Oxford University Press, New York, 2002.