Building Secure SRAM PUF Key Generators on Resource Constrained Devices

Yansong Gao*†, Yang Su‡, Wei Yang*, Shiping Chen†, Surya Nepal†, and Damith C. Ranasinghe‡
*School of Computer Science and Engineering, Nanjing University of Science and Technology, China
†Data61, CSIRO, Syndey, Australia
‡Auto-ID lab, School of Computer Science, The University of Adelaide, Adelaide, Australia
yansong.gao@njust.edu.cn; yang.su01@adelaide.edu.au; generalyzy@gmail.com; shiping.chen@data61.csiro.au;
surya.nepal@data61.csiro.au; damith.ranasinghe@adelaide.edu.au

***Abstract*—A securely maintained key is the premise upon which data stored and transmitted by ubiquitously deployed resource limited devices, such as those in the Internet of Things (IoT), are protected. However, many of these devices lack a secure non-volatile memory (NVM) for storing keys because of cost constraints. Silicon physical unclonable functions (PUFs) offering unique device specific secrets to electronic commodities are a low-cost alternative to secure NVM. As a physical hardware security primitive, reliability of a PUF is affected by thermal noise and changes in environmental conditions; consequently, PUF responses cannot be directly employed as cryptographic keys. A fuzzy extractor can turn noisy PUF responses into usable cryptographic keys. However, a fuzzy extractor is not immediately mountable on (highly) resource constrained devices due to its implementation overhead. We present a methodology for constructing a lightweight and secure PUF key generator for resource limited devices. In particular, we focus on PUFs constructed from pervasively embedded SRAM in modern microcontroller units and use a batteryless computational radio frequency identification (CRFID) device as a representative resource constrained IoT device in a case study.**

***Index Terms*—SRAM PUF, Reverse fuzzy extractor, Multiple reference response, PUF key generator, Computational RFID**

I. Introduction

Resource limited IoT (Internet of Things) devices provide challenging environments for building privacy and security preserving mechanisms. Although various cryptographic algorithms can be engineered to address the aforementioned challenges, all of these measures eventually rely on a securely maintained key. Nowadays, digital keys are stored in non-volatile memory (NVM) such as FLASH that is often assigned externally to a computing platform. However, it has been shown that securing digital key storage is non-trivial in practice due to e.g., technical or cost limitations [1]. Silicon physical unclonable functions (PUFs) [2], [3] provide an alternative. PUFs can replace the functionality of the NVM based keys while increasing the security level of cryptographic key storage using standard CMOS technology based hardware security primitives. Overall, silicon PUFs offer: i) low fabrication costs [1]; ii) uniqueness (an inseparable *fingerprint* of a hardware instance); and iii) enhanced resistance to attacks, especially, invasive attacks [4]–[6].

Physical Unclonable Functions (PUFs): A PUF, in essence, extracts secrets from inevitable process variations that is alike device fingerprints [2], [8], [9]. Hence, in reality, identical PUF instances cannot be forged, not even by the same manufacturer. As illustrated in Fig. 1(a), a PUF can be abstracted as a function where a given binary input (challenge $\mathbf{c}$) applied to different PUF instances produces differing instance-specific binary outputs (responses $\mathbf{r}$). According to the number of challenge response pairs (CRPs) yielded, a PUF is generally categorized into strong and weak PUF classes [10]. Generally, a strong PUF yields CRPs exponential with respect to the size of the challenge or area. Conversely, its counterpart, weak PUF, yields limited number of CRPs—CRP space linearly increases with the area or the size of the challenge. Elementary PUF applications are: i) lightweight authentication; and ii) key generation.

Lightweight Authentication: PUFs provide novel avenues for realizing lightweight authentication mechanisms, usually employing strong PUFs due to the infeasibility of fully characterizing all its CRPs in a reasonable time frame by an adversary [10]. In the PUF enrolment phase, the server (*verifier*) lodges a number of CRPs into its database. In the authentication phase when the PUF integrated device (*prover*) is deployed in field, the server randomly picks up a challenge and sends it to the prover. Once the prover returns the measured response subject to the received challenge, the server compares the enrolled response in the database with the returned response from the prover. The authenticity of the prover is established only if the Hamming distance between the received and enrolled response is less than a predetermined threshold. However, due to various model building attacks [11]–[14], it is now recognized that it is hard to provide security guarantees

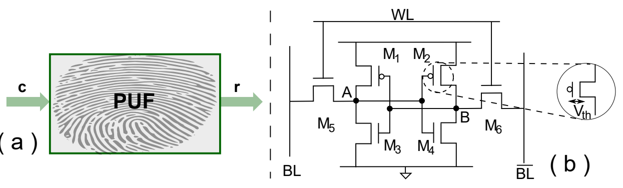

Fig. 1. (a) Queried by a challenge, the PUF produces an instance-specific response. (b) SRAM PUF: Threshold voltage $V_{\rm th}$ mismatches of the transistors determines the response [7]. For example, when the $V_{\rm th,M_1}$ is slightly smaller than $V_{\rm th,M_2}$, at power-up, the transistor $\rm M_1$ starts conducting before $\rm M_2$, thus, A = '1'. This in turn prevents $\rm M_2$ switching on. As a consequence, the SRAM power-up state prefers to be '1' (A = '1', B ='0'), which is the response, while the address is the challenge.

for simple challenge-response based lightweight authentication protocols built upon silicon strong PUFs, specifically, variants of the APUF [15] and k-sum ROPUF [16]. One method to halt modeling attacks is to limit the exposed number of CRPs by limiting the number of authentication rounds. The trade-off is that the PUF has to be disposed once a predetermined number of authentication rounds are reached [17]. A recent examination of strong PUF based authentication mechanisms concluded that a secure PUF based authentication mechanism is better to be crafted from a PUF derived key [14], [18]. *Consequently, we will focus on key generation with PUFs and how secure key generation can be realized on resource limited devices*. A PUF based key derivation method has the potential to provide resource limited devices a cost effective authentication mechanism together with secure key storage.

Secure and Reliable Key Generation: Unlike strong PUF responses that are usually correlated, weak PUF responses are normally independent from each other where each response is derived from an independent source of entropy. Consequently, a weak PUF, in general, is inherently immune to modeling attacks. Weak PUFs are highly suitable for deriving cryptographic keys. Recall that PUFs measure physical circuit properties to generate responses and—like any physical measurement—are inevitably affected by thermal noise and varying environmental conditions; hence, PUF response reproductions are not completely stable [1]. Therefore, a PUF based key generator utilizes a fuzzy extractor to turn unstable responses into a stable and uniformly distributed cryptographic key. PUF based key generators have been well studied in the last decade, however, only small number of studies provide a complete implementation. Among the full implementations, majority are hardware implementations on FPGA platforms [19], [20] rather than software implementations on microcontroller (MCU) platforms [21]. In fact, most IoT devices do not have FPGAs but operate using MCUs. Further, to the best of our knowledge, there is only one end-to-end software PUF key generator implementation on a low power computing platform in the form of a highly resource constrained and batteryless device that relies only on harvested RF energy for operations [22].*Therefore, we aim to provide a simple and timely guide to the Ubiquitous Computing community for realizing PUF based key generators to benefit from their inherent key security.* In particular:

- We consider the use of freely available SRAM on low cost and low power MCUs to function as an intrinsic PUF to derive a secure and reliable cryptographic key.
- We consider a methodology for realizing lightweight implementations of reliable key generators capable of execution on low cost and ultra low power microcontrollers.

II. Anatomy of a Key Generator

A. SRAM PUF

SRAM memory is pervasively embedded within electronic commodities. When SRAM is powered up, each SRAM cell has a favored power-up state; see Fig. 1(b). Such a favored power-up state varies from cell to cell, and chip to chip. Therefore, the power-up pattern of SRAM memory can be treated as a PUF where the address of each cell is a challenge and power-up state the response. Thus SRAM yields an intrinsic PUF attributing to its wide scale availability on MCUs and its ability to be used without any overhead—extra hardware [7]; this makes SRAM PUFs a popular silicon PUFs nowadays.

B. (Reverse) Fuzzy Extractor

To turn the raw noisy PUF responses—for example, from an SRAM PUF—into a cryptographic key, a widely accepted approach is to employ a fuzzy extractor [20], [23], [24]—see the illustration in Fig. 2. A fuzzy extractor consists of: i) a secure sketch; and ii) a procedure for entropy extraction.

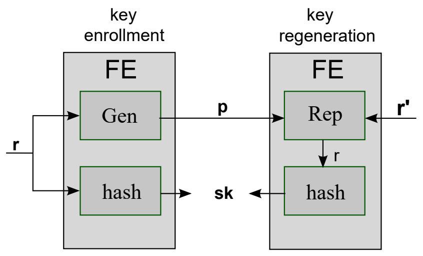

Fig. 2. During the key enrollment phase, the Gen() takes enrolled response $\mathbf{r}$ as input, and computes helper data $\mathbf{p}$. The enrolled secret key $\mathbf{sk}$=hash($\mathbf{r}$). During the key regeneration phase, the Gen() takes both the regenerated response $\mathbf{r}'$ and the helper data $\mathbf{p}$ as inputs, and recovers the enrolled response $\mathbf{r}$. The recovered $\mathbf{r}$ is hashed to obtain the enrolled secret key $\mathbf{sk}$.

In general, the secure sketch construction has a pair of functions: Gen() and Rep()—see overall illustration in Fig. 2. There are two prevalent secure sketch constructions: i) code-offset construction; and ii) syndrome construction [23]. We choose the later, mainly considering its security—see Section V. During key enrollment phase, helper data $\mathbf{p}$ is computed by using Gen($\mathbf{r}$), where $\mathbf{p} = \mathbf{r} \times \mathbf{H^T}$ and $\mathbf{H}$ is a parity check matrix of a linear error correction code. Key reconstruction described by Rep($\mathbf{r}'$,$\mathbf{p}$)—where $\mathbf{r}'$ is the reproduced response that may be slightly different from the enrolled response $\mathbf{r}$—first constructs a syndrome $\mathbf{s} = (\mathbf{r}' \times \mathbf{H^T}) \oplus \mathbf{p} = \mathbf{e} \times \mathbf{H^T}$, with $\mathbf{e}$ an error vector. Then through an error location algorithm, $\mathbf{e}$ is determined. Subsequently, the response $\mathbf{r}$ is recovered through $\mathbf{r} = \mathbf{e} \oplus \mathbf{r}'$. The recovered PUF response $\mathbf{r}$ may not be uniformly distributed. Therefore, an entropy extraction step, using a universal hash function for example, is employed to generate a cryptographic key with full bit entropy.

In a fuzzy extractor setting, the Gen() function is performed by the server during the enrollment phase to compute helper data while the Rep() function is implemented on the field deployed token. By recognizing that the computational burden of the Rep() function is significantly more than that of Gen() function, Van Herrewege *et al.* [19] placed Gen() on the resource-constraint token while leaving the computationally heavy Gen() function execution to the resource-rich server; this arrangement is termed the reverse fuzzy extractor (RFE).

The (R)FE must bear two properties: i) **Correctness**: It must be possible to correctly reconstruct $\mathbf{r}$—equivalently, the secret key $\mathbf{sk}$—-given helper data $\mathbf{p}$ if the Hamming distance between the enrolled response $\mathbf{r}$ and the reevaluated response $\mathbf{r}'$ is less than an acceptable threshold; and ii) **Security**: Given exposed helper data $\mathbf{p}$, there must be adequate residual entropy in the PUF response $\mathbf{r}$ to ensure the security of the key $\mathbf{sk}$.

Although the entropy extractor realized with a hash function can be lightweight, the overhead of the secure sketch is comparatively more dominant. *This is a significant problem PUF based key generation on a resource constrained device, often limited by computational capability, memory, and power.* We will focus on the correct recovery in the context of resource limited devices and a RFE where the Gen() function is implemented on the resource limited token in Section III and IV. We discuss security in Section V.

III. Reducing Reverse Fuzzy Extractor Overhead

A. Response Pre-Processing

The overhead of implementing the Gen() function is directly related to the response unreliability or bit error rate (BER); noisier responses naturally require a higher error correction overhead. Therefore, it is imperative to reduce the expected BER of PUF responses on a token. In this context, response pre-processing methodologies, e.g., majority voting and reliable response pre-selection [25] can be employed.

In majority voting (MV), given q repeated response measurements under an operating condition, the response value ('1'/'0') enrolled is that which shows no less than $\lceil \frac{q}{2} \rceil$ measurements; with q an odd integer. In reliable response pre-selection (PreSel), each PUF response under an operating condition is repeatedly measured i times, only response bits that exhibit 100% reliable regeneration (all '1's/'0's) are selected for use and enrollment—a detailed methodology with a resource limited device is in [22].

B. Key Generation with Multiple Reference Responses

Gao *et al.* recently proposed the multiple reference responses (MRR) enrollment strategy [26]. This method is especially suitable in a reverse fuzzy extractor setting to significantly reduce the Gen() function implementation overhead on a token [26]. The MRR rationale is described below in the context of a reverse fuzzy extractor, dubbed MR^{3}FE.

Fig. 3 illustrates MR^{3}FE in the context of key generation. During the enrollment phase, in contrast to conventional single response enrolment under a nominal operating condition, e.g., room temperature of 25°C, the server enrolls J reference responses $\{\mathbf{r}_1, ..., \mathbf{r}_j, ...\mathbf{r}_J\}$ subject to the same challenge $\mathbf{c}$ applied to the same PUF but under J different operating conditions. We take two enrolled reference responses as an example. As shown in Fig. 4, we assume that the server enrolls two reference responses, $\mathbf{r}_1$ and $\mathbf{r}_2$, evaluated under 50°C and 0°C, respectively. We can observe in Fig. 4—regardless of the operating condition under which the reference response is enrolled—when the difference between the reference operating condition at enrollment increases with respect to the condition during a regenerated response, the the expected BER increases.

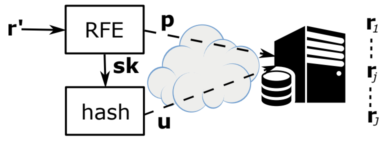

Fig. 3. MR^{3}FE in the context of key generation on token.

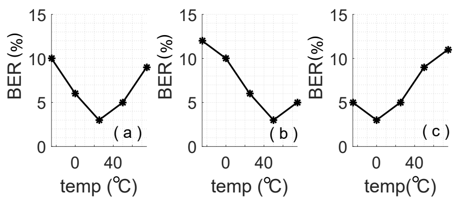

Fig. 4. Bit error rate when the reference response is evaluated under: (a) 25°C; (b) 50°C; and (c) 0°C.

When the PUF is deployed in the field, as shown in Fig. 3, the token reevaluates $\mathbf{r}'$ corresponding to a challenge $\mathbf{c}$, typically stored on chip—recall that we are using a reverse fuzzy extractor. Then the token computes helper data $\mathbf{p} \leftarrow$Gen($\mathbf{r}'$) and generates secret key $\mathbf{sk}$ during run-time. The token sends $\mathbf{p}$ and $\mathbf{u} \leftarrow$hash($\mathbf{sk}||\mathbf{p}$) to the server—$\mathbf{u}$ allows the server to check its success of recovering $\mathbf{sk}$, while also ensuring the integrity of the helper data $\mathbf{p}$ to alleviate potential helper data manipulation attacks [23], [27]. The server now attempts to recover the response $\mathbf{r}'$ based on two enrolled reference responses: $\mathbf{r}_1$ and $\mathbf{r}_2$. For example, the server first uses $\mathbf{r}_1$ to restore $\mathbf{r}'' \leftarrow$Rep($\mathbf{r}_1$, $\mathbf{p}$). Then the server extracts a secret key $\mathbf{sk}' \leftarrow$hash($\mathbf{r}''$) and consequently computes verification value $\mathbf{u}'$=hash($\mathbf{sk}'||\mathbf{p}$). If $\mathbf{u}' = \mathbf{u}$, then it implies that the $\mathbf{r}''$=$\mathbf{r}'$, equivalently, the $\mathbf{sk}$ is deemed to be correctly restored. Otherwise, $\mathbf{sk}$ regeneration fails based on $\mathbf{r}_1$. The server continues to use reference response $\mathbf{r}_2$ to try to regenerate $\mathbf{sk}$. Consequently, if any of the enrolled reference responses are capable of recovering $\mathbf{sk}$, MR^{3}FE is deemed successful and both the token and the server are in possession of the secret key $\mathbf{sk}$.

MRR Advantages: We provide an intuitive illustration to explain. Let us firstly assume that we use the *conventional single reference response* enrolled under 25°C as shown in Fig. 4. We further assume that the regenerated response is evaluated under −25°C where the expected BER is 10% and suppose that the helper data $\mathbf{p}$ is only capable of correcting upto 6% of response errors. It is clear that the RFE using reference response evaluated under 25°C to recover the secret key will most likely fail. Now given two reference responses $\mathbf{r}_1$ and $\mathbf{r}_2$ evaluated under 50°C and 0°C, respectively. We can see that $\mathbf{r}_2$ is more likely to be able to recover the secret key $\mathbf{sk}$ even though the helper data $\mathbf{p}$ still has the same 6% error correction capability because the expected BER at −25°C using $\mathbf{r}_2$ as reference is only 5%. What if the regenerated response $\mathbf{r}'$ is evaluated under 80°C? It is not difficult to see that now $\mathbf{r}_1$ is highly likely to successfully recover the secret key $\mathbf{sk}$. Although the server is unable to control the operating condition under which the response $\mathbf{r}'$ is regenerated, we can observe that the server now has the capability of choosing one of multiple reference responses to ensure that the chosen reference response is close to the operating condition under which the $\mathbf{r}'$ is regenerated. *This ability greatly relaxes the error correction requirement and, thus, the implementation overhead of the* Gen() *function.*

C. Key Failure Rate

Our study uses the family of BCH(n, k, t) linear codes with a syndrome based decoding strategy to realize a reverse fuzzy extractor considering its popularity [20], [23] and its security [23], [26], [27]. Here, n is the codeword length, k is the code size, t is the number of errors that can be corrected within this n-bit block. Assuming response bit errors are independently and identically distributed (i.i.d.), we can express the average key failure rate of recovering an n-bit response $\mathbf{r}'$ based on a selected reference response $\mathbf{r}_j$, termed as $\mathbb{P}_{1j}$, with $j \in \{1, .., J\}$ with J the number of multiple references employed by the server, as:

$$\mathbb{P}_{1j} = 1 - \mathsf{F}_B(t; n, \mathrm{BER}_j) \tag{1}$$

where BER_j is the expected BER using $\mathbf{r}_j$ as the reference response. Here, $\mathsf{F}_B()$ is a cumulative density function of a binomial distribution with t successes in n Bernoulli trials, with each trial having success probability of BER_j.

A BCH(n, k, t) encoding produces $(n-k)$-bit helper data that will be publicly known while the k bits contribute to the secret. For a single BCH(n, k, t) block, the complexity of finding the k-bit extracted key from the n-bit response $\mathbf{r}'$ is 2^k. It is not common to use a single large BCH(n, k, t) block; typically a large block is split into small processing blocks to reduce implementation complexity [28]. For k bits of key material, response $\mathbf{r}'$ can be divided into multiple non-overlapping blocks of a BCH(n_1, k_1, t_1) code where $n_1 < n$ and $k_1 < k$ for a parallel implementation. Now the complexity of finding the k bit secret is $2^{k_1 \cdot L}$ where L is the number of parallel BCH(n_1, k_1, t_1) code blocks used to realize k bits of secret key material. Given a BCH(n_1, k_1, t_1) code employed to gain a security level of k bits with $L = \lceil k/k_1 \rceil$ blocks, the key recovery failure rate under the assumption of i.i.d code blocks is:

$$\mathbb{P}_{2j} = 1 - (1 - \mathbb{P}_{1j})^L. \tag{2}$$

When all J reference responses $\{\mathbf{r}_1, ..., \mathbf{r}_j, ..., \mathbf{r}_J\}$ are used, $\mathbf{r}'$ reconstruction fails only when *all* reference responses cannot restore the response $\mathbf{r}'$. We adopt a very conservative evaluation of the key failure rate $\mathbb{P}_{\text{fail}}$ expressed as[1]:

$$\mathbb{P}_{\text{fail}} = min\{\mathbb{P}_{2j}\}, \; j \in \{1, ..., J\} \tag{3}$$

[1]For a detailed discussion, the reader is referred to [26].

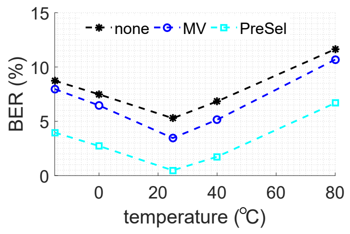

Fig. 5. BER when reference response is enrolled under 25°C.

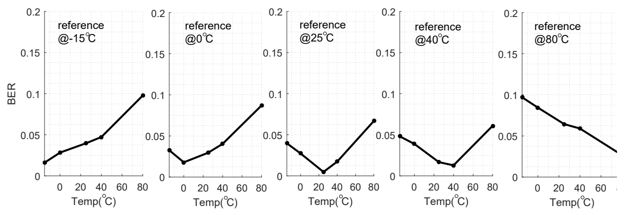

Fig. 6. BER when a reference response is enrolled under different operating conditions.

IV. Experimental Validations

We first test the reliability of embedded SRAM PUFs. Then the software implementation set-up and the means of assessing clock cycle overhead is detailed. Finally, we implement the MR^{3}FE PUF key generator and validate the significant overhead reduction realized in comparison with: i) a single reference response based FE; and ii) a single reference response based RFE.

A. SRAM PUF Evaluation

1) SRAM PUF Dataset: The PUF CRP dataset used is from 23 MSP430FR5969 microconrollers (MCUs) embedded in CRFID transponders. From each MCU, we read power-up states of 16,384 (2 KB) SRAM cells as SRAM PUF responses. The SRAM PUF reliability is much less sensitive to voltage variations compared with temperature fluctuations attributing to the SRAM cell's symmetric structure [3], [29]. Hence, we focus on its reliability under varying temperature conditions: $-15°$C, $0°$C, $25°$C, $40°$C and $80°$C. Under each temperature condition, each response bit is repeatedly measured 100 times.

2) Bit Error Rate: As shown in Fig. 5, for single reference response enrollment, BER under three different enrollment strategies are applied: one-time response measurement (none), majority voting (MV) with 9 repeated measurements ($q = 9$), and preselection (PreSel) with 10 repeated measurements. The PreSel outperforms others.

Fig. 6 illustrates the BER when MRR are enrolled under differing operating conditions. For the reference response enrolled under $25°$C, preselection is applied to select reliable responses. Then reference response under $-15°$C, $0°$C, $40°$C and $80°$C applies response majority voting on the pre-selected reliable responses.

B. Overhead Evaluation

1) Testing Setup: The test environment used is Texas Instruments' (TI) Code Composer Studio (CCS) 7.2.0, the C code used is downloaded to a MSP430FR5969 LaunchPad Evaluation Kit via USB. TI CCS has a built-in GCC toolchain for our hardware kit. This includes the `msp430-gcc-6.4.0.32` win32 compiler. The software instructions are executed sequentially as advanced out-of-order execution is unavailable for typical resource-constraint MCUs. The overhead of software implementation is assessed in terms of clock cycles to complete the algorithm—this is our primary evaluation measure. We measured clock cycles using Profile Clock tool supported in the CCS environment.

Two key components of the PUF key generator are the hash function and the BCH code encoding/decoding—Gen()/Rep()—blocks. Based on the hash function evaluations in [22], we selected BLAKE2s-128 as it showed the best performance, in term of clock cycles. As for BCH(n_1, k_1, t_1) code, it is imperative to use a small n_1 and t_1. First, the BCH(n_1, k_1, t_1) code overhead decreases sharply when the codeword length n_1 decreases. Second, a small t_1 implies a correspondingly large k_1. With a large k_1, to achieve the security level of k-bit secret key $\mathbf{sk}$, the needed number of BCH(n_1, k_1, t_1) blocks—$L = \lceil k/k_1 \rceil$—will reduce, thus, reducing the overall overhead to gain k-bit entropy for the secret key $\mathbf{sk}$.

2) Overhead Result and Comparison:

- **Fuzzy Extractor with Single Reference Response:** To achieve $\mathbb{P}_{\text{fail}} < 10^{-6}$, nine BCH(127,15,27) blocks are required giving $\mathbb{P}_{\text{fail}} = 1.66 \times 10^{-7}$ when the single reference response under 25°C is used. One BCH(127,15,27) block *decoding* consumes 2,102,222 clock cycles. The fuzzy extractor needs to sequentially execute BCH(127,15,27) decoding blocks 9 times and two BLACKE2s-128 hash operations. In total, it consumes up to 19,127,446 clock cycles.
- **Reverse Fuzzy Extractor with Single Reference Response:** To achieve $\mathbb{P}_{\text{fail}} < 10^{-6}$, nine BCH(127,15,27) blocks are required giving $\mathbb{P}_{\text{fail}} = 1.66 \times 10^{-7}$ when the single reference response under 25°C is used. *Notably, the token only needs to perform the encoding operation requiring significantly less clock cycles in comparison with decoding*. One BCH(127,15,27) block *encoding* consumes 111,335 clock cycles. This reverse fuzzy extractor needs to sequentially execute BCH(127,15,27) encoding 9 times and two BLACKE2s-128 hash operations. In total, it consumes up to 1,211,461 clock cycles.
- **Reverse Fuzzy Extractor with Three Reference Response:** To achieve $\mathbb{P}_{\text{fail}} < 10^{-6}$, eight smaller BCH(63,16,11) blocks are adequate to meet the requirement with $\mathbb{P}_{\text{fail}} = 9.85 \times 10^{-7}$. Here, 3MRR under -15°C, 25°C, 80°C are used. One BCH(63,16,11) block *encoding* consumes 51,003 clock cycles. This reverse fuzzy extractor needs to sequentially execute BCH(63,16,11) encoding 8 times and two BLACKE2s-128 hash operations. In total, it consumes up to 617,470 clock cycles.

We can see that the reverse fuzzy extractor is especially suitable for resource limited devices attributing to the fact that the Gen overhead is significantly less than the Rep overhead. The reverse fuzzy extractor overhead is only 6.3% of the fuzzy extractor when a single reference response is deployed to achieve the same key failure rate performance. When the MRR is adopted by the reverse fuzzy extractor, the clock cycle overhead is further reduced by 49% in comparison with a single reference response. This is because smaller n_1 and t_1 code parameters—meaning less encoding overhead—-of a BCH(n_1,k_1,t_1) code can achieve the targeted key failure rate. It is worth mentioning here that the server has the ability to enroll more reference responses under fine-grained operating conditions to further reduce the MR^{3}FE overhead. Although this creates a *one-time* burden on the server, the impact on the token through the reduction in overhead costs is significant and the benefits extend to the lifetime of the token in the field.

V. Security Discussion

Given a secure sketch with BCH(n,k,t) code, entropy leakage is caused mainly from the public helper data. The well-known min-entropy loss is the $n-k$ bound given the exposure of helper data. This is mainly caused by the $(n-k)$-bit helper data exposure. Extra entropy leakage can be from response bias [30], [30], [31]; a bias exists when the probability of a response being '1' or '0' is not ideally 50%. Active helper data manipulation (HDM) attacks may also cause serious entropy leakage if the helper data algorithm is not carefully constructed [23], [27], [32], [33]. More specifically, not all error correction codes and decoding strategies that are components of helper data algorithm for implementing the fuzzy extractor are able to guarantee the security of the PUF derived key [23], [27], therefore, when a helper data algorithm is introduced for realizing a fuzzy extractor, not only its implementation overhead but also its security against helper data manipulation attacks has to be carefully evaluated. Nonetheless, Our MR^{3}FE case study employed BCH codes and syndrome decoding, which has been shown to be secure under HDM attacks [27].

The reverse fuzzy extractor can result in unanticipated entropy loss under repeated helper data exposure associated with a given PUF response $\mathbf{r}'$; unless, PUF responses are unbiased. Generally, the extra entropy loss is a result of the leakage of bit-specific reliability information [31]. However, the above extra entropy losses are important only when PUF response bias is considerably different from the ideal value of 50% as shown by the analysis in [18], [31]. In practice, modern silicon PUFs usually have a good bias performance—close to 50% [3]. The bias of our tested SRAM PUF is 49.87%. Therefore, the extra entropy loss is very small. In this context, employing a few more response bits in the reverse fuzzy extractor can compensate for the small extra loss in entropy. As observed by Delvaux [18], for a PUF with low bias within $[0.42, 0.58]$, increasing the length of raw responses alone is an effective measure.

If the bias is severe, entropy compensation by solely increasing the length of raw responses becomes ineffective. As a result, debiasing the biased raw responses [30] must be undertaken first, e.g., via classic von Neumann (CVN) debiasing, pair-output von Neumann debiasing with erasures (ϵ-2O-VN) and Hamming Weight (HW) based de-biasing [22]—an *example* of employing a HW based debiasing method for key derivation with a resource limited devices is detailed in [22]. Notably, not all debiasing schemes offer reusability—multiple use of the same PUF response—for a reverse fuzzy extractor [31]. Thus, debiasing schemes, e.g., ϵ-2O-VN, that offer reusability should be chosen for the reverse fuzzy extractor.

VI. Conclusion

We studied a reverse fuzzy extractor based PUF key derivation method with a focus on reducing the token implementation overhead. The overhead is reduced by reducing the response unreliability; realized by two compatible methodologies: i) reliable response preselection; and ii) multiple reference response enrollment performed during the PUF provisioning (enrollment) phase. For experiments, we used the intrinsic SRAM PUF—requires neither hardware modification nor extra area cost—to demonstrate a lightweight PUF key generator implementation on a battery-less CRFID device, followed by security analyses.

VII. Acknowledgement

We acknowledge support from Australian Research Council Discovery Program (DP140103448), NJUST Research Start-Up Funding (AE89991/039, AE89991/103) and the National Natural Science Foundation of China (61802186).

References

[1] M. Hiller, “Key derivation with physical unclonable functions,” Ph.D. dissertation, Universität München, 2016.

[2] G. E. Suh and S. Devadas, “Physical unclonable functions for device authentication and secret key generation,” in *Proc. Design Automation Conf. (DAC)*. ACM, 2007, pp. 9–14.

[3] M. Roel, “Physically unclonable functions: Constructions, properties and applications,” Ph.D. dissertation, University of KU Leuven, 2012.

[4] P. Tuyls, G.-J. Schrijen, B. Škorić, J. Van Geloven, N. Verhaegh, and R. Wolters, “Read-proof hardware from protective coatings,” in *Cryptographic Hardware and Embedded Systems-CHES 2006*. Springer, 2006, pp. 369–383.

[5] B. Gassend, M. V. Dijk, D. Clarke, E. Torlak, S. Devadas, and P. Tuyls, “Controlled physical random functions and applications,” *ACM Transactions on Information and System Security*, vol. 10, no. 4, p. 3, 2008.

[6] J. Obermaier, V. Immler, M. Hiller, and G. Sigl, “A measurement system for capacitive PUF-based security enclosures,” in *Proceedings of the 55th Annual Design Automation Conference*. ACM, 2018, p. 64.

[7] Holcomb, Daniel E, W. P. Burleson, and K. Fu, “Power-up SRAM state as an identifying fingerprint and source of true random numbers,” *IEEE Trans. Comput.*, vol. 58, no. 9, pp. 1198–1210, 2009.

[8] C.-H. Chang, Y. Zheng, and L. Zhang, “A retrospective and a look forward: Fifteen years of physical unclonable function advancement,” *IEEE Circuits and Systems Magazine*, vol. 17, no. 3, pp. 32–62, 2017.

[9] Y. Gao, H. Ma, S. F. Al-Sarawi, D. Abbott, and D. C. Ranasinghe, “PUF-FSM: A controlled strong PUF,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 64, pp. 2532–2543, 2017.

[10] C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas, “Physical unclonable functions and applications: A tutorial,” *Proc. IEEE*, vol. 102, pp. 1126–1141, 2014.

[11] U. Ruhrmair, J. Solter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, and S. Devadas, “PUF modeling attacks on simulated and silicon data,” *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 11, pp. 1876–1891, 2013.

[12] G. T. Becker, “The gap between promise and reality: On the insecurity of XOR Arbiter PUFs,” in *Cryptographic Hardware and Embedded Systems (CHES)*. Springer, 2015, pp. 535–555.

[13] G. T. Becker, “On the pitfalls of using Arbiter-PUFs as building blocks,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst*, vol. 34, no. 8, pp. 1295–1307, 2015.

[14] J. Delvaux, “Machine learning attacks on PolyPUF, OB-PUF, RPUF, and PUF–FSM,” *IACR Cryptology ePrint Archive*, 2017.

[15] Y. Gao, G. Li, H. Ma, S. F. Al-Sarawi, O. Kavehei, D. Abbott, and D. C. Ranasinghe, “Obfuscated challenge-response: A secure lightweight authentication mechanism for PUF-based pervasive devices,” in *IEEE International Conference on Pervasive Computing and Communication Workshops*, 2016, pp. 648–653.

[16] M.-D. Yu, D. M’Raihi, I. Verbauwhede, and S. Devadas, “A noise bifurcation architecture for linear additive physical functions,” in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2014, pp. 124–129.

[17] M.-D. Yu, M. Hiller, J. Delvaux, R. Sowell, S. Devadas, and I. Verbauwhede, “A lockdown technique to prevent machine learning on PUFs for lightweight authentication,” *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 2, no. 3, pp. 146–159, 2016.

[18] J. Delvaux, “Security analysis of PUF-based key generation and entity authentication,” Ph.D. dissertation, University of KU Leuven and ShangHai Jiao Tong University, 2017.

[19] A. Van Herrewege, S. Katzenbeisser, R. Maes, R. Peeters, A.-R. Sadeghi, I. Verbauwhede, and C. Wachsmann, “Reverse fuzzy extractors: Enabling lightweight mutual authentication for PUF-enabled RFIDs,” in *Financial Cryptography and Data Security*. Springer, 2012, pp. 374–389.

[20] R. Maes, A. Van Herrewege, and I. Verbauwhede, “PUFKY: A fully functional PUF-based cryptographic key generator,” in *Cryptographic Hardware and Embedded Systems*. Springer, 2012, pp. 302–319.

[21] A. Aysu, E. Gulcan, D. Moriyama, P. Schaumont, and M. Yung, “End-to-end design of a PUF-based privacy preserving authentication protocol,” in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2015, pp. 556–576.

[22] Y. Su, Y. Gao, M. Chesser, O. Kavehei, A. Sample, and D. C. Ranasinghe, “SecuCode: Intrinsic PUF Entangled Secure Wireless Code Dissemination for Computational RFID Devices,” *ArXiv e-prints*, Jul. 2018.

[23] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede, “Helper data algorithms for PUF-based key generation: Overview and analysis,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 6, pp. 889–902, 2015.

[24] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, “Fuzzy extractors: How to generate strong keys from biometrics and other noisy data,” *SIAM journal on computing*, vol. 38, no. 1, pp. 97–139, 2008.

[25] C. Böhm and M. Hofer, “PUF with preselection,” in *Physical Unclonable Functions in Theory and Practice*. Springer, 2013, pp. 239–248.

[26] Y. Gao, Y. Su, L. Xu, and D. C. Ranasinghe, “Lightweight (reverse) fuzzy extractor with multiple reference puf responses,” *IEEE Transactions on Information Forensics and Security*, 2018.

[27] G. T. Becker, “Robust fuzzy extractors and helper data manipulation attacks revisited: Theory vs practice,” *IEEE Transactions on Dependable and Secure Computing*, 2017, DOI: 10.1109/TDSC.2017.2762675.

[28] M. Hiller, M.-D. Yu, and G. Sigl, “Cherry-picking reliable PUF bits with differential sequence coding,” *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 9, pp. 2065–2076, 2016.

[29] G. Selimis, M. Konijnenburg, M. Ashouei, J. Huisken, H. de Groot, V. van der Leest, G.-J. Schrijen, M. van Hulst, and 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*. IEEE, 2011, pp. 567–570.

[30] R. Maes, V. van der Leest, E. van der Sluis, and F. Willems, “Secure key generation from biased PUFs: extended version,” *Journal of Cryptographic Engineering*, vol. 6, no. 2, pp. 121–137, 2016.

[31] J. Delvaux, D. Gu, I. Verbauwhede, M. Hiller, and M.-D. M. Yu, “Efficient fuzzy extraction of PUF-induced secrets: Theory and applications,” in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2016, pp. 412–431.

[32] J. Delvaux and I. Verbauwhede, “Key-recovery attacks on various ro PUF constructions via helper data manipulation,” in *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2014, p. 72.

[33] J. Delvaux and I. Verbauwhede, “Attacking PUF-based pattern matching key generators via helper data manipulation,” in *Topics in Cryptology–CT-RSA*. Springer, 2014, pp. 106–131.