X-Lock
A Secure XOR-Based Fuzzy Extractor for Resource Constrained Devices

Liberati, Edoardo; Visintin, Alessandro; Lazzeretti, Riccardo; Conti, Mauro; Uluagac, Selcuk

DOI
10.1007/978-3-031-54770-6_8

Publication date
2024

Document Version
Final published version

Published in
Applied Cryptography and Network Security - 22nd International Conference, ACNS 2024, Proceedings

X-Lock: A Secure XOR-Based Fuzzy Extractor for Resource Constrained Devices

Edoardo Liberati[1(✉)], Alessandro Visintin[2], Riccardo Lazzeretti[1], Mauro Conti[2,3], and Selcuk Uluagac[4]

[1] Sapienza University of Rome, Rome, Italy
e.liberati@diag.uniroma1.it
[2] University of Padua, Padua, Italy
[3] Delft University of Technology, Delft, Netherlands
[4] Florida International University, Miami, FL, USA

Abstract. The Internet of Things rapid growth poses privacy and security challenges for the traditional key storage methods. Physical Unclonable Functions offer a potential solution but require secure fuzzy extractors to ensure reliable replication. This paper introduces X-Lock, a novel and secure computational fuzzy extractor that addresses the limitations faced by traditional solutions in resource-constrained IoT devices. X-Lock offers a reusable and robust solution, effectively mitigating the impacts of bias and correlation through its design. Leveraging the preferred state of a noisy source, X-Lock encrypts a random string of bits that can be later used as seed to generate multiple secret keys. To prove our claims, we provide a comprehensive theoretical analysis, addressing security considerations, and implement the proposed model. To evaluate the effectiveness and superiority of our proposal, we also provide practical experiments and compare the results with existing approaches. The experimental findings demonstrate the efficacy of our algorithm, showing comparable memory cost ($\approx$ 2.4 KB for storing 5 keys of 128 bits) while being 3 orders of magnitude faster with respect to the state-of-the-art solution (0.086 ms against 15.51 s).

Keywords: Fuzzy extractor · Physical Unclonable Functions · Error tolerant cryptography

1 Introduction

Within the domain of Internet of Things (IoT), the presence of resource-constrained devices poses significant obstacles in ensuring the development of robust privacy and security mechanisms. While numerous cryptographic algorithms can be customized to address these challenges, their effectiveness hinges on the availability of securely maintained keys. Presently, prevalent practices involve the storage of digital keys in a Non-Volatile Memory (NVM), typically situated externally to the computing platform. However, the practical realization of secure digital key storage has emerged as a formidable endeavor, primarily attributed to technical limitations or cost-related considerations [14].

In this context, Physical Unclonable Functions (PUFs) [20] arise as a promising alternative solution. PUFs represent a viable replacement for NVM-based keys and offer several notable advantages, including cost-effectiveness, inherent uniqueness, and heightened resistance against various attacks [18]. PUFs derive confidential information from inherent process variations, similar to distinctive device fingerprints [7]. The replication of identical PUF instances becomes then practically unattainable, even when originating from the same manufacturer. Nevertheless, PUFs rely on physical circuit properties for generating responses, making them susceptible to factors such as thermal noise and environmental conditions. Consequently, achieving reliable replication of PUF responses poses a significant challenge [14].

The notion of fuzzy extractor [10] emerged as a highly regarded approach for addressing key management issues associated with error-prone data. A fuzzy extractor offers the ability to extract an identical random string from a noisy source without the need to store the string itself, therefore allowing the usage of noisy sources such as PUFs as cryptographic primitives. The general construction comprises two algorithms. The generation algorithm *Gen* takes an initial string S_{read} of the noisy source as input and produces a string K along with a helper data H. Subsequently, the reproduction algorithm *Rep* leverages the helper data H to reproduce the string K from a second string S'_{read} of the same source, given that the distance between S_{read} and S'_{read} is sufficiently small. The correctness property of a fuzzy extractor ensures that the reproduction process yields the same string K when the fuzzy data is generated from the same source. Additionally, the security aspect of a fuzzy extractor guarantees that the helper data H does not divulge any information about the original fuzzy data.

Previous works employed secure sketches and randomness extractors as core components for their constructions [2,25,26]. A secure sketch is an information reconciliation protocol that enables the recovery of the original S_{read} from a received S'_{read} when they are sufficiently close. Subsequently, a random string is extracted from S_{read} using a randomness extractor. However, it has been observed that secure sketches leak information through the helper data, leading to a loss of security [13]. Consequently, the construction of a fuzzy extractor based on secure sketches necessitates the utilization of high entropy source data.

Recently, a novel class of solutions introduced an innovative approach to construct fuzzy extractors [5,6,13,28]. These solutions employ S_{read} to encrypt a random string in a manner that allows for decryption with knowledge of a closely related string S'_{read}. Fuller et al. [13] presented a computational fuzzy extractor based on the Learning With Errors (LWE) problem. Their approach effectively mitigates information leakage from the helper data by concealing secret randomness within it. However, this construction exhibits notable inefficiency and only tolerates sub-linear errors. Additionally, it lacks guarantees of reusability (see Definition 2) and robustness (see Definition 3). Canetti et al. [5,6] employed digital lockers as their cryptographic primitive. Their solution is applicable even with low-entropy sources and ensures reusability. However, their sample-then-lock technique necessitates excessive storage space for the helper values. Woo et al. [28] proposed a solution based on non-linear LWE, which offers memory efficiency while simultaneously guaranteeing both reusability and robustness. Nevertheless, their solution is computationally demanding and may not be suitable for restricted environments.

This paper presents X-Lock, a novel design for a secure and cost-effective computational fuzzy extractor. Similar to [5,6,13,28], X-Lock utilizes the noisy strings obtained from a fuzzy source to protect a random string of bits. X-Lock encrypts each bit of the random string by XOR-ing it multiple times with a subset of bits from the noisy string. Through this approach, X-Lock offers reusability and robustness while inherently addressing bias and correlation issues within the fuzzy source. Bias quantifies the equilibrium between 0 s and 1 s within the fuzzy response, whereas correlation evaluates the degree of independence among distinct bits within the same fuzzy response. To substantiate these assertions, we conducted a comprehensive theoretical analysis encompassing both the security and the implementation aspects of our model. Furthermore, we conducted practical experiments and comparisons with existing state-of-the-art approaches to demonstrate the superiority of our proposal.

The empirical results demonstrate the optimal performance of our proposed model. With a key size of 128 bits, our algorithm uses ≈ 2.4 KB to store 5 keys. Furthermore, the execution time for the *Rep* procedure amounts to 0.086 ms. In contrast, Canetti et al. [5,6] necessitates 216.20 MB and ≈ 1 min computational time, while Woo et al. [28] utilizes ≈ 2.8 KB and operates at a speed of 15.51 s (see Sect. 6). Overall, our model achieves comparable memory cost with the state-of-the-art solutions while outperforming them by being 3 orders of magnitude faster.

Contributions. In summary, this paper offers the following contributions:

- We present X-Lock, a novel design for a secure and cost-effective fuzzy extractor based on XOR operations;
- We introduce the XOR iteration (XOR-ation) as our cryptographic primitive that efficiently mitigates source bias and correlation;
- We provision our model with robustness, reusability and insider security through straightforward and effective strategies;
- We provide a thorough theoretical analysis of our model including storage, computational and security aspects;
- We implement X-Lock in C language and conduct practical experiments to compare our results to the current state-of-the-art solutions.

Organization. The rest of the paper is organized as follows. In Sect. 2 we introduce previous works and the state-of-the-art in reusable and robust fuzzy extractors. In Sect. 3 we provide background information on PUFs, fuzzy extractors, bias and correlation. In Sect. 4 we describe the algorithm of X-Lock in details. In Sect. 5 we explain the rationale and conduct a security analysis over our methodology. In Sect. 6 we describe our implementation of the algorithm and show the results of the comparison with both Canetti et al. [5,6] and Woo et al. [28]. Finally, Sect. 7 closes the paper with some final remarks.

2 Related Works

Since the pioneering work of Dodis et al. [10], the concept of fuzzy extractor has emerged as a prominent solution for managing keys derived from noisy sources. Most fuzzy extractors adopt the sketch-then-extract paradigm, employing secure sketches and randomness extractors as fundamental components [2,25,26]. However, secure sketches entail a minor data leakage from the helper data, leading to compromised security [13]. Consequently, the utilization of a secure sketch-based fuzzy extractor necessitates high-entropy source data.

The notion of reusability, as formalized by Boyen [2], pertains to the security of multiple pairs of extracted strings and associated helper data, even when such helper data is exposed to an adversary. The work showcases that achieving information-theoretic reusability requires a significant reduction in security, thereby implying an inherent trade-off. Related again to fuzzy extractors security, Boyen et al. [3] introduce the notion of a robust fuzzy extractor to safeguard the helper data against malicious modifications, ensuring the detection of any such alterations. Both these security criteria must be satisfied to ensure the viability of fuzzy extractors as secure authentication methods in real-life scenarios.

In a departure from secure sketch-based approaches, Fuller et al. [13] propose a computational fuzzy extractor based on the LWE problem. Their methodology employs noisy strings to encrypt a random string in such a way that it can be decrypted with the knowledge of another closely related string. By concealing secret randomness within the helper data instead of extracting it from the noisy string, their approach mitigates data leakage concerns. Nonetheless, their construction exhibits significant inefficiency in terms of memory requirements and computational time and can only tolerate sub-linear errors, while lacking reusability. Apon et al. [1] improve upon Fuller's construction to fulfill the reusability requirement by utilizing either a random oracle or a lattice-based symmetric encryption technique. However, their solution remains unable to overcome the limitations of the construction by Fuller et al. [13], allowing only a logarithmic fraction of error tolerance. Another proposal by Wen et al. [25] presents a reusable fuzzy extractor based on the LWE problem, resilient to linear fractions of errors. Nevertheless, their scheme relies on a secure sketch and consequently results in the leakage of sensitive data.

Addressing the limitations of prior works, Canetti et al. [5,6] introduce a fuzzy extractor construction employing inputs from low-entropy distributions, leveraging the concept of digital lockers [4,16] as a symmetric key cryptographic primitive. Their strategy involves sampling multiple partial strings from a noisy input string, hashing them independently, and then locking individual secret keys with each hashed partial string. Correctness ensues if there is a high likelihood of successfully recovering at least one hashed subset of data upon a second measurement. To achieve this, the scheme necessitates a large helper data size for storing the hashed values and employs an inefficient reproduction algorithm. To address this concern, Cheon et al. [8] made adjustments to the first scheme proposed by Canetti et al. [5], reducing the size of the helper data. However, this modification results in an increased computational cost for the reproduction algorithm due to the introduction of a significant amount of hashing operations.

A recent contribution by Woo et al. [28] presents a novel computational fuzzy extractor that does not rely on a secure sketch or digital lockers. Their construction offers security under the non-linear LWE problem and encodes the extracted key using two cryptographic primitives: error correcting codes (ECCs) and the EMBLEM encoding method [21]. This innovative approach achieves both robustness and reusability while tolerating linear errors. However, it assumes that the source data is drawn from a uniform distribution and, although more efficient than previous works, it still entails a non-negligible computational effort that may limit its adoption on low-end devices.

In contrast with previous works, X-Lock efficiently satisfies all the security properties required from a secure fuzzy extractor. The protocol only uses XOR operations in its procedure, thus cutting the computational complexity. Additionally, the XOR primitive inherently introduces a mitigation for both source bias and correlation issues (see Sect. 3). It is noteworthy that none of the previous works adequately addresses such bias and correlation issues. In real-world setups, fuzzy sources rarely produce perfectly random outputs and may exhibit biases towards a specific value, as well as correlations among bits. While the construction by Canetti et al. [5,6] implicitly addresses the correlation problem, the bias issue remains unaddressed in the existing literature. Security guarantees of proposals in the existing literature are summarized in Table 1.

Table 1. Properties comparison.

Construction	Correctness	Leak Prevention	Reusability	Robustness	Correlation	Bias
Canetti et al. [5,6]	●	●	●	●	●	
Fuller et al. [13]	●	●				
Apon et al. [1]	●	●				
Wen et al. [25]	●		●			
Cheon et al. [9]	●	●	●	●	●	
Woo et al. [28]	●	●	●	●		
Our work	●	●	●	●	●	●

3 Background

General Notation. We use lowercase letters to denote single values. We use capital letters to denote groups of values that are either organized in sets or strings. With $\{\cdot\}^x$ we denote a group of cardinality x that contains values from those described inside the braces. We denote with $|X|$ the cardinality of the group X. We define $||X||$ as the Hamming weight of X, i.e. the number of its non-zero values. The notation $Pr[\cdot]$ expresses the probability of the event described inside the square brackets. We denote the XOR logical operation with $\oplus$. When applied to sequences of bits, $\oplus$ is intended to perform a bit-wise XOR. We introduce the XOR-ation operator $\bigoplus_{i=0}^{I}$, performing the XOR operation iteratively over a sequence of bits denoted by the indexes $0 \leq i \leq I$. Note that the order of the indexes does not change the final result. We provide a summary of the notation used throughout the paper in Table 2. As a final remark, notice that we adapted definitions coming from the literature according to our notation, for sake of clarity and coherence.

Table 2. Notation summary.

Symbol	Meaning
$\|X\|$	Cardinality of group X
$\|\|X\|\|$	Hamming weight of group X
$\bigoplus_{y \in Y}$	XOR-ation over the set of indexes Y
F	Fuzzy source
S_{read}, S_{pref}	Random string and preferred string of a fuzzy source
e_{rel}, e_{abs}	Relative error and absolute error
Gen, Rep	Procedures involved in a fuzzy extractor
K, H	Secret key and helper data generated by Gen
p_{bias}	Number of 1s in a string over its total number of bits
ϕ	Correlation factor of a fuzzy source
B	Poll of random bits
C	Bits involved in the XOR-ation
L	Bit-locker, group of locks relative to the same bit
K_{pre}	Final values before obtaining K
V	Vault
n	nonce
R	Set of random indices of original bits constituting a key
T	Authentication token
$\perp$	None value

PUFs Characterization. PUFs leverage the inherent random variations introduced during the silicon manufacturing process to generate secret keys on the fly [20]. These variations, akin to unique fingerprints, serve as a cost-effective source of randomness. Conceptually, a PUF can be viewed as a function where a specific binary input (*i.e.,* a challenge) elicits distinct binary outputs (*i.e.,* responses) specific to each individual PUF instance. However, outputs obtained from the same PUF circuit may exhibit variations due to factors such as thermal noise and environmental conditions.

Consequently, we conceptualize a PUF circuit as a fuzzy source F that provides a string of bits S_{read} upon request. When comparing two such sequences from F they can exhibit different values in certain positions. We formalize this aspect with the relative error $e_{rel} = \frac{||S_{read} \oplus S'_{read}||}{|S_{read}|}$, that is the ratio of bits in S_{read} that differ from S'_{read}. Together with S_{read}, we introduce the concept of preferred string S_{pref}, which is obtained by identifying the value appearing most frequently (*i.e.,* more than 50% of the times) for each bit within the string. We calculate it by using a statistically significant number of S_{read} (*e.g.,* more than 100 strings). Figure 1 reports the estimation of the number of bits we expect to be incorrectly identified with respect to the number of strings read to craft S_{pref}. The plot shows both the trend of the expected number of errors, which exponentially decreases as the number of strings increases, and the threshold on the number of strings that brings such expectation below 1. Similarly to e_{rel}, we define the absolute error e_{abs} as the ratio of bits in S_{read} that differ from S_{pref}. Notably, the two errors are linked by the inequality $0 \leq e_{rel} \leq 2 \cdot e_{abs}$. Indeed, consider two strings S_{read} and S'_{read} that have wrong bits in the same positions, if any. Then their relative error is $e_{rel} = 0$. On the other hand, consider two strings S_{read} and S'_{read} that exhibit the maximum possible error e_{abs} with respect to S_{pref} but have no wrong shared bit. Then their relative error is $e_{rel} = 2 \cdot e_{abs}$.

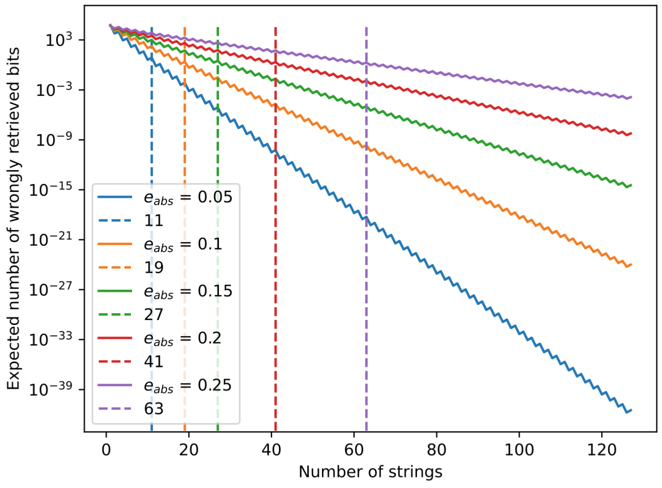

Fig. 1. Expected number of wrong bits (logarithmic scale) over number of strings. (Color figure online)

It is crucial to underscore that when setting up the fuzzy extractor using the raw S_{read} instead of S_{pref}, it is likely to introduce additional errors into the process, ultimately increasing the likelihood of recovering an incorrect key. Hence, in the context of this paper, we employ S_{pref} as our reference string and, consequently, we consider e_{abs} to be the error rate associated with F.

Fuzzy Extractors. Fuzzy extractors [10] are cryptographic constructions designed to derive reliable and uniformly distributed cryptographic keys from sources prone to noise or errors. This process involves two probabilistic procedures. The *Gen* procedure takes a string S_{read} and produces a random string K along with a public helper string H. If another string S'_{read} is sufficiently close to S_{read} (*i.e.*, $||S_{read} \oplus S'_{read}|| \leq t$ for a small t), the *Rep* procedure can utilize the helper string H to correctly reproduce the original random string K from S'_{read}.

In this work, we focus on computational fuzzy extractors that consider the scenario where potential attackers possess knowledge about the noise distribution and have control over the errors. To formalize this concept, we define a metric space as a finite set $\mathcal{M}$ equipped with a distance function $dis : \mathcal{M} \times \mathcal{M} \rightarrow R^{+} = [0, \infty)$ that satisfies the identity property ($dis(x, y) = 0$ if and only if $x = y$), the symmetry property ($dis(x, y) = dis(y, x)$), and the triangle inequality ($dis(x, z) \leq dis(x, y) + dis(y, z)$). The statistical distance [5,6] $SD(X, Y)$ is defined as $\frac{1}{2} \sum_x (Pr[X = x] - Pr[Y = x])$.

Definition 1 (Computational fuzzy extractor, [12] Definition 4). *Given a metric space $(\mathcal{M}, dis)$, let $\mathcal{F}$ be a family of probability distributions over $\mathcal{M}$. A pair of randomized procedures (Gen,Rep) is an $(\mathcal{M}, \mathcal{F}, k, t, \epsilon)$-computational fuzzy extractor with error δ if Gen and Rep satisfy the following properties:*

- *Gen on input $S_{read} \in \mathcal{M}$ outputs $K \in \{0,1\}^k$ and a helper string $H \in \{0,1\}^*$.*
- *Rep takes $S'_{read} \in \mathcal{M}$ and $H \in \{0,1\}^*$ as inputs.*
- ***Correctness.** If $dis(S_{read}, S'_{read}) \leq t$ and $(K, H) \leftarrow Gen(S_{read})$, then $Pr[Rep(S'_{read}, H) = K] \geq 1 - \delta$, where the probability is over the randomness of (Gen, Rep). If $dis(S_{read}, S'_{read}) > t$, then no guarantee is provided about the output of Rep.*
- ***Security.** For any $F \in \mathcal{F}$, K is pseudo-random conditioned on H, that is if $(K, H) \leftarrow Gen(S_{read})$ then $SD((K, H), (U_k, H)) \leq \epsilon$.*

The correctness property guarantees that the fuzzy extractor can accurately retrieve the protected data when provided with an input that is sufficiently close to the original. The security property ensures that the output of the fuzzy extractor does not reveal any specific information about the underlying noisy distribution.

To utilize a computational fuzzy extractor as an authentication mechanism, it must possess both reusability and robustness. Reusability [2] is related to the case where the helper data associated to several extracted strings is revealed to an adversary.

Definition 2 (Reusability, [6] Definition 6). *Let $\mathcal{F}$ be a family of distributions over $\mathcal{M}$. Let $(F^1, F^2, \ldots, F^\rho)$ be ρ correlated random variables such that $\forall i = 1, \ldots, \rho : F^i \in \mathcal{F}$. Let (Gen, Rep) be a $(\mathcal{M}, \mathcal{F}, k, t, \epsilon)$-computational fuzzy extractor with error δ. Let D be a distinguisher that outputs 0 when it believes that the input was randomly produced and 1 when it believes it was produced by (Gen, Rep). Define the following game $\forall j = 1, \ldots, \rho$:*

- ***Sampling.*** *The challenger samples* $S^j_{read} \leftarrow F^j$ *and* $u \leftarrow \{0,1\}^k$.
- ***Generation.*** *The challenger computes* $(K^j, H^j) \leftarrow Gen(S^j_{read})$.
- ***Distinguishing.*** *The advantage of* D *is quantifiable as*
  $Adv(D) = Pr[D(K^1, \ldots, K^{j-1}, K^j, K^{j+1}, \ldots, K^\rho, H^1, \ldots, H^\rho) = 1] - Pr[D(K^1, ..., K^{j-1}, u, K^{j+1}, ..., K^\rho, H^1, ..., H^\rho) = 1]$.

(Gen, Rep) *is* (ρ, σ)*-reusable if* $\forall D \in \mathcal{D}$ *and* $\forall j = 1, \ldots, \rho$, *the advantage is at most* σ.

In an intuitive sense, if a fuzzy extractor is reusable, it implies that a specific key K remains secure even when the adversary possesses knowledge of all associated helper data. This includes both the helper data linked to the key K and the helper data related to all other keys. In other words, this property ensures the security of keys generated by the fuzzy extractor even when the generation process is repeated multiple times across different strings, consequently permitting the reuse of the same secret noisy source (*e.g.,* the same iris, the same fingerprint, the same PUF, etc.) in multiple contexts.

As underlined in Canetti et al. [5,6], the key aspect is that the family of distributions $\mathcal{F}$ is arbitrarily correlated, meaning no assumption is made on the correlation between the distributions.

Robustness [3] addresses the scenario where an adversary modifies the helper data H before it is given to the user. A robust fuzzy extractor ensures that any changes made by an adversary to H will be detected.

Definition 3 (Robustness, [3] Definition 6). *Let* $\mathcal{F}$ *be a family of distributions over* $\mathcal{M}$. *Let* $\perp$ *denote that Rep detected a tampered output. Let* (Gen, Rep) *be a* $(\mathcal{M}, \mathcal{F}, k, t, \epsilon)$*-computational fuzzy extractor with error* δ. *Let* $(K, H) \leftarrow Gen(S_{read})$, *with* S_{read} *output of* F. *Let* A *be an adversary, and* $H' \leftarrow A(K, H)$ *with* $H \neq H'$. *Then* (Gen, Rep) *is a* τ*-robust fuzzy extractor if* $Pr[Rep(S'_{read}, H') \neq \perp] \leq \tau$.

Bias and Correlation in Fuzzy Sources. In real-world scenarios, fuzzy sources often exhibit non-uniform distributions in their output bit strings. These deviations from uniformity can arise from two main factors: bias and correlation among the physical components of the source. When bias is present [17], it means that either 0-bits or 1-bits occur more frequently than the other. We quantify bias by the parameter p_{bias}, with $p_{bias} \cdot l$ being the expected number of 1-bits in a l-bit response. An unbiased source would have $p_{bias} = 0.5$, indicating an equal probability for 0-bits and 1-bits. A biased (non-uniform) source, instead, exhibits unbalanced quantities of 1-bits and 0-bits, offering the adversary an advantage when trying to compromise the source itself. Correlation, on the other hand [27], refers to the lack of independence among the values of the bits generated by the noisy source. This lack of independence arises due to physical dependencies, such as cross-talk noise among Static Random Access Memory (SRAM) cells [19], where the value of one bit can be influenced by neighboring bits in the physical medium. Such relationship among bits significantly decreases the security guarantees of the system since disclosing a bit may compromise several other bits, reducing the overall search space size. To precisely define the concept of correlation, we introduce the notion of correlation classes of bits.

Definition 4 (Correlation class of a bit). *Let $F \in \mathcal{F}$ be a fuzzy source drawn from a family of noisy sources $\mathcal{F}$. Let $\sim$ denote the equivalence relation such that, given two bits b and b', the value of a bit can be inferred by knowing the value of the other one if $b \sim b'$. The equivalence relation exhibits the commutative property, i.e., $b \sim b'$ if and only if $b' \sim b$. Let S_{pref} be the preferred string of F. Then, the correlation class $[b]$ of bit b is the set $\{b' \in S_{pref} : b' \sim b\}$ of elements that are equivalent to b.*

It is important to note that the correlation classes are disjoint, meaning that a bit can only belong to a single class. The total number of correlation classes corresponds to the maximum number of independent values that can be extracted from a particular source. In this work, we assume that the correlation classes have approximately the same size, meaning that each independent value has an equal probability of being selected in a random draw. To quantify the level of correlation, we introduce the correlation factor ϕ.

Definition 5 (Correlation factor). *Let $F \in \mathcal{F}$ be a fuzzy source drawn from a family of noisy sources $\mathcal{F}$. Let S_{pref} be the preferred string of F. Let $[S_{pref}]$ denote the set of all correlation classes in S_{pref} and $|[S_{pref}]|$ the total number of correlation classes in S_{pref}. Then, the correlation factor is defined as $\phi = 1 - \frac{|[S_{pref}]|}{|S_{pref}|}$.*

The correlation factor provides an immediate measure of dependency between the source bits.

For example, a correlation factor of $\phi = 0.75$ indicates that we expect three dependant values for every four bits in the source.

Threat Model. The model involves two parties, a legit user and an attacker. The user leverages the PUF to extract multiple secure keys out of the noisy source. The user may be either a human owning a device or the device itself. Instead, the attacker aims at compromising the extracted keys. In this scenario, we consider an adversary who has complete control over the communication channels [11]. The adversary has access to the appliance that executes the algorithm and stores its data. However, we explicitly exclude the possibility of extensive physical attacks or invasive side-channel attacks by the adversary. These types of attacks would require unrestricted access to the device for extended periods of time, which would raise suspicion and allow for their detection. Furthermore, we assume that the adversary can read data from standard non-volatile memory and modify it. However, the PUF in the system possesses a tamper-evidence property. If the adversary attempts to learn the secret stored in it, the behavior of the PUF will change significantly or be destroyed, thereby indicating the tampering. Lastly, we assume that all algorithms related to our protocol are public, but implemented in a way that prevents modification. This assumption ensures that the adversary cannot tamper with or modify the algorithms to their advantage. Overall, our focus is to design a theoretically secure protocol which ensures reliable replication of a key out of a PUF. Other potential vulnerabilities are then out of the scope of this work.

4 X-Lock: Construction Details

We consider a noisy source F that outputs a bit-string S_{read} of length $|S_{read}|$ upon request. We denote S_{pref} as the preferred state of F and e_{rel}, e_{abs} to be respectively its relative and absolute errors. Section 3 provides a thorough description of these quantities.

Similarly to existing works [5,6,13,28], our algorithm shares a common thread in leveraging the outputs generated by F to encrypt a random string of bits B. However, we significantly differentiate from the previous proposals as we employ S_{pref} to encrypt a large pool of bits B by means of the XOR-ation operator. We then decrypt and combine random subsets of B to generate multiple secret keys. The core element of the algorithm is the vault V resulting from the encryption of B. We provide a reference of the structure of V in Fig. 2. It consists of a sequence of bit-lockers, which individually guard the encryption of a single bit $b \in B$. Each bit-locker is a collection of locks L, where each lock is the XOR of a distinct b with the XOR-ation of a random subset of bits from S_{pref}. To restore the bit value b is then sufficient to retrieve a S_{read} from F and perform the XOR-ation on the lock value with same subset of bits. In Sect. 5 we show that this construction indeed forms a valid computational fuzzy extractor.

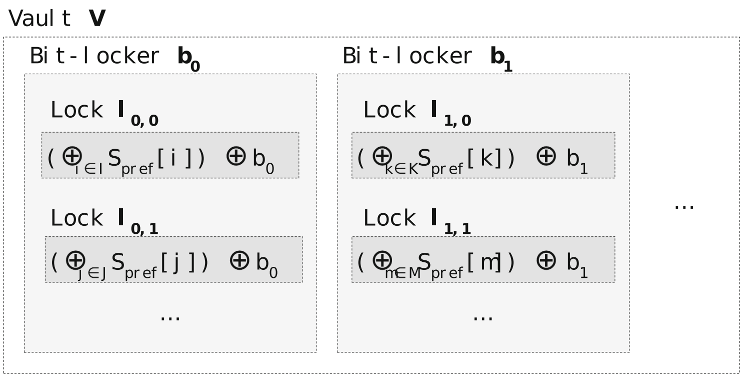
Vault **V**
Bit-locker **b0**
Lock **l0,0**
$(\bigoplus_{i\in I} S_{pref}[i]) \oplus b_0$
Lock **l0,1**
$(\bigoplus_{j\in J} S_{pref}[j]) \oplus b_0$
...
Bit-locker **b1**
Lock **l1,0**
$(\bigoplus_{k\in K} S_{pref}[k]) \oplus b_1$
Lock **l1,1**
$(\bigoplus_{m\in M} S_{pref}[m]) \oplus b_1$
...
...

Fig. 2. Composition of the vault V. It is a collection of bit-lockers, each referring to a specific bit $b \in B$. A bit-locker contains multiple locks that XOR the bit of reference with the XOR-ation of a random subset of bits from the source F.

To perform the initial encryption of B into V, our algorithm introduces an *Init* procedure that needs to be executed only once.

Subsequently, our approach involves the two standard probabilistic procedures *Gen* and *Rep*. In the following, we provide a thorough description of the three procedures, integrated by the pseudo-code of the design presented in Algorithm 1.

Init. The *Init* procedure (line 1) is responsible for generating the initial vault V.

The procedure receives in input the preferred string S_{pref}, the random pool of bits B, the number of locks $|L|$ per bit-locker, and the number of bits $|C|$ to use in each XOR-ation. We provide details on the generation of S_{pref} in Sect. 3. We first iterate over all the bits $b \in B$ (line 3), and for each b we generate $|L|$ locks (line 5). For each lock l, we select $|C|$ random indexes (line 6) and use them to select the bits in S_{pref} for the XOR-ation (line 7). The procedure eventually returns V (line 10).

The function $drawVaultIndexes$ (line 6) selects $|C|$ indexes spanning from 1 to $|S_{pref}|$ without replacement, *i.e.,* with no index selected more than once. We provide details of its implementation in Sect. 5.3.

The use of multiple locks adds correctness to our construction. We demonstrate this aspect in Sect. 5.1. Additionally, the XOR-ation mitigates the effects of bias and correlation in F. We provide an explanation in Sect. 5.2.

Gen. The *Gen* procedure (line 11) generates a novel key K, the nonce n, the set of indexes R and its authentication token T. Notice that $H' = (n, R, T)$ constitutes part of the helper data $H = (V, H')$. In particular, H' is the portion of helper data specifically related to each key, whilst V is common and shared among all keys.

The procedure takes in input the string S_{read}, the vault V, the number of bits $|B|$ of the random pool B, and the number of bits $|K|$ of the key K. We first generate a random nonce n (line 12) and a sequence R of $|K|$ random indexes in the range $[1, |B|]$ (line 13). The indexes $r \in R$ represent the subset of bits decrypted from the vault V to form the key K. To achieve this, we perform an operation that is specular to the one executed in the *Init* procedure. For each $r \in R$ (line 15), we access all the lock values and index lists (l, C) (line 17). For each (l, C), we XOR l with the XOR-ation of S_{read} over C and collect the resulting bit b' in X. The final bit value specific to r is then obtained as the most common value in X and stored into K_{pre} (line 20). To protect the restored values, the final value of the key K is generated via hashing $K_{pre}$with the nonce n (line 21). We produce a token T as the hash of key K with the list of indexes R.

The generation of T adds robustness to the methodology. We discuss it in further detail in Sect. 5.1. The procedure returns the key K, the nonce n, the list of indexes R, and the token T.

The function $drawKeyIndexes$ (line 13) selects $|K|$ indexes spanning from 1 to $|B|$ without replacement. We provide more details in Section 5.3. Notice that $drawVaultIndexes$ and $drawKeyIndexes$ could be implemented with the same procedure since both draw a certain number of indexes spanning in a certain range without replacement.

Rep. The *Rep* procedure (line 24) restores the key K from the vault V.

The procedure receives in input the string S'_{read} and the helper data $H = (V, H') = (V, (n, R, T))$, where V is the vault, n is the nonce, R is the list of indexes, and T is the authentication token. The procedure first iterates over $r \in R$ and restores the majority values similarly to the *Gen* procedure (lines 26 - 31). It then generates the key K' (line 32) and the token T' (line 33). In case T' does not coincide with T, the procedure returns a null value (line 35). Otherwise, it outputs the restored key K' as valid (line 36).

5 X-Lock: Algorithm Analysis

5.1 Security Analysis

We first introduce two definitions that are propaedeutic to the security analysis.

Definition 6 (Common elements between combinations). *Let X be a group of elements with cardinality $|X|$. Consider the process of drawing a random combination of y elements without replacement out of X. Then the probability p_{share} for two combinations to share at most z elements is quantifiable as*

$$p_{share} = \sum_{i=0}^{z} \frac{\binom{y}{i}\binom{|X|-y}{y-i}}{\binom{|X|}{y}}.$$

The numerator computes the number of combinations that share exactly i elements. It selects i elements out of the y available, followed by choosing the remaining $y - i$ elements out of the remaining $|X| - y$ values. Dividing by the total number of available combinations yields the partial percentage. The summation then sums all the contributions up to z common elements.

Definition 7 (Odd binomial distribution). *Let consider a Bernoulli trial with probability p of success and y independent trials. Then the odd binomial distribution considers the contributions of odd indexes in the summation of trial probabilities. In formula,*

$$p_{odd} = \sum_{i=0}^{\frac{y}{2}} \binom{y}{2i+1} \cdot p^{2i+1} \cdot (1-p)^{y-2i-1}.$$

The formula employs Bernoulli trials to assess the probability of having exactly $2i + 1$ successes in the elements. The summation from 0 to $y/2$ and the index $2i + 1$ permit to consider all the odd numbers between 0 and y.

We then introduce the Roucé-Capelli theorem that stands at the core of our security demonstration.

Theorem 1 (Rouché-Capelli). *A system of linear equations with n variables has a solution if and only if the rank of its coefficient matrix A is equal to the rank of its augmented matrix $A|b$. If there are solutions, they form an affine subspace $\mathbb{R}^n$ of dimension $n - rank(A)$. In particular:*

Algorithm 1. X-Lock algorithm. [] denotes an empty array. X[y] denotes the value at index y in array X. [x,y] denotes the interval of numbers from x to y.

```
1: procedure $Init(S_{pref}, B, |L|, |C|)$
2:     $V \leftarrow []$
3:     for $b \in B$ do
4:         $L \leftarrow []$
5:         for _ in $[1, |L|]$ do
6:             $C \leftarrow$ drawVaultIndexes($[1, |S_{pref}|]$, $|C|$)
7:             $l \leftarrow (\bigoplus_{c \in C} S_{pref}[c]) \oplus b$
8:             $L$.append$(l, C)$
9:         $V$.append$(L)$
10:    return $V$
11: procedure $Gen(S_{read}, V, |B|, |K|)$
12:    $n \leftarrow$ getNonce()
13:    $R \leftarrow$ drawKeyIndexes($[1, |B|]$, $|K|$)
14:    $K_{pre} \leftarrow []$
15:    for $r \in R$ do
16:        $X \leftarrow []$
17:        for $(l, C) \in V[r]$ do
18:            $b' \leftarrow (\bigoplus_{c \in C} S_{read}[c]) \oplus l$
19:            $X$.append$(b')$
20:        $K_{pre}$.append(getMajorityValue$(X)$)
21:    $K \leftarrow hash(K_{pre}, n)$
22:    $T \leftarrow hash(K, R)$
23:    return $K, n, R, T$
24: procedure $Rep(S'_{read}, V, n, R, T)$
25:    $K'_{pre} \leftarrow []$
26:    for $r \in R$ do
27:        $X \leftarrow []$
28:        for $(l, C) \in V[r]$ do
29:            $b' \leftarrow (\bigoplus_{c \in C} S'_{read}[c]) \oplus l$
30:            $X$.append$(b')$
31:        $K'_{pre}$.append(getMajorityValue$(X)$)
32:    $K' \leftarrow hash(K'_{pre}, n)$
33:    $T' \leftarrow hash(K', R)$
34:    if $T \neq T'$ then
35:        return $\perp$
36:    return $K'$
```

– *if $n = rank(A)$, then the solution is unique;*
– *otherwise there are infinitely many solutions.*

Proof. See introductory level book on linear algebra and geometry [22]. □

The Rouché-Capelli theorem states that a system of linear equations possesses a unique solution only if the number of independent equations equals the number of total variables. If there are more equations than variables, the system may become inconsistent, leading to the absence of a solution. Conversely, if there are fewer equations, the system becomes under-determined, providing an infinite number of potential solutions.

We are now ready to provide the proof of the security of the vault V.

Theorem 2 (Solution space of the vault). *Let S_{pref} be the preferred string of F. Let V be a vault with $|L|$ locks per bit-locker and $|C|$ bits per XOR-ation. Let ϕ be the correlation factor related to F. Then the solution space of V has at least dimension $(1 - \phi) \cdot |S_{pref}| + (1 - |L|) \cdot |B|$.*

Proof. The pool vault V is a collection of bit-lockers, each composed of $|L|$ locks that guard an individual bit from B (see Fig. 2). Each lock XORs a distinct bit from B with a XOR-ation of $|C|$ bits from S_{pref}. It is easy to see that a lock is a linear equation, thus making the vault V a system of $|B| \cdot |L|$ equations. Given the correlation factor ϕ, the independent variables provided by F to the system are $|S_{pref}| - \phi \cdot |S_{pref}| = (1 - \phi) \cdot |S_{pref}|$. B provides exactly $|B|$ variables as its bits are independent by definition. Thus the number of variables in the system is $(1 - \phi) \cdot |S_{pref}| + |B|$. We then recall from Theorem 1 that the dimension of the solution space is calculated as $n - rank(A)$, with n the total number of involved variables. In our case, $rank(V) \leq |B| \cdot |L|$ as some of the equations may not be independent. Hence we end up with the inequality $n - rank(V) \geq ((1 - \phi) \cdot |S_{pref}| + |B|) - (|B| \cdot |L|)$, thus proving the statement. □

The solution space of the vault is directly related to the security of the protocol, as it determines the number of parameters a potential attacker needs to provide to solve the linear system. A closer look to Theorem 2 suggests that, for increasing the security of the vault V, we want high $|S_{pref}|, |B|$ and low $\phi, |L|$.

Definition 8 (Security of the vault). *A vault V is (α, β)-secure if the number of total locks equals α and the probability for two locks to share at most half of the elements is less than β.*

By imposing that at most half of the elements are dependent, we make sure that combining two equations results in an equation with more variables than the initial ones. We provide experimental evidence of this in Sect. 6. The definition permits to manage the security of the vault V. Notice that the dimension of the random pool B is determined by the two parameters α, β. As per Theorem 2, $\phi, |S_{pref}|, |L|, |B|$ determine the solution space. $\phi, |S_{pref}|$ are determined by the physical parameters of the fuzzy source F. We can then set $|L|$ and require the vault V to provide a certain probability that mixing locks does not provide information to the attacker. By doing this, we set the desired solution space and thus the dimension of $|B|$.

We can now show that Algorithm 1, described in Sect. 4, satisfies the interpretation of computational fuzzy extractor given in Definition 1.

Theorem 3. *X-Lock is a $(\mathcal{M}, \mathcal{F}, k, t, \epsilon)$-computational fuzzy extractor with error δ.*

Proof. We first notice that the helper data for a particular key in our construction is $H = (V, (n, R, T))$. The vault V is generated once and shared by all keys, while $H' = (n, R, T)$ is specific for each key. Following Definition 1, we first show that (Gen, Rep) are valid functions for a computational fuzzy extractor. Procedure Gen takes in input $S_{read}, V, |B|, |K|$ and outputs K, n, R, T. This aligns with the definition $(K, H) \leftarrow Gen(S_{read})$, where $V, |B|, |K|$ are supporting input parameters and $H = H' = (n, R, T)$ for the output. Procedure Rep takes in input S'_{read}, V, n, R, T and outputs either K' or $\perp$. This again conforms to the general definition $K \leftarrow Rep(S'_{read}, H)$, with $H = (V, (n, R, T))$ in the input and $K = K'$ for the output.

We now proceed in demonstrating the correctness of the algorithm. Specifically, we show that the procedure correctly retrieves the keys with distance t between strings measured through e_{abs} and retrieval error δ proportional to $e_{bitlock}$. We focus on a single bit-locker, as the generalization is trivial. Recalling from Sect. 4, a bit-locker is a collection of locks that XOR a bit $b \in B$ with a XOR-ation of bits from S_{pref}. By using multiple incorrect bits the probability of having an error augments. Considering the absolute error e_{abs}, we can calculate the probability e_{lock} for a lock to return the wrong value by using the odd binomial distribution in Definition 7. In particular, we set the probability of success $p = e_{abs}$ and the number of trials $y = |C|$. This formulation captures the idea that a XOR-ation returns an incorrect value whenever there is an odd number of errors in its elements. Using e_{lock}, we can now estimate the probability $e_{bitlock}$ for a bit-locker to reconstruct an incorrect bit:

$$e_{bitlock} = \sum_{i=\frac{|L|}{2}}^{|L|} \binom{|L|}{i} \cdot {e_{lock}}^{i} \cdot (1 - e_{lock})^{|L|-i} \tag{1}$$

The final bit is calculated by majority voting. Therefore, the procedure restores the wrong bit value whenever the number of errors is greater than half of the total elements $|L|$. We use a Bernoulli trial to calculate probability of having exactly i errors, and use a summation to aggregate all the results from $\frac{|L|}{2}$ to $|L|$. To achieve the desired level of correctness we can tune the number of locks $|L|$ to meet a given error. The same procedure is valid for all bit-lockers, thus the final error is proportional to $e_{bitlock}$, hence satisfying the initial claim.

We eventually provide proof of security of the construction. To satisfy Definition 1, we need to show that the key K appears random even if a potential attacker possesses knowledge of the helper data $(V, (n, R, T))$. It is important to note that both K and T are derived using a cryptographic hash function, where $K \leftarrow hash(K_{pre}, n)$ and $T \leftarrow hash(K, R)$. We assume that the hash function exhibits the typical secure properties (*i.e.,* pre-image resistance, second pre-image resistance, and collision resistance). Considering all these properties together, the hash function generates outputs that appear to be random. Therefore, the pseudo-randomness of the resulting key K is conditioned solely on the amount of information leaked by the helper data $(V, (n, R, T))$. $H' = (n, R, T)$ does not leak any information on the underlying vault V: n is a random nonce,

R is a random collection of indexes, and T is computed through a hash function. Regarding the vault V, we previously showed that it is (α, β)-secure under Definition 8. The parameters (α, β) can be used to tune ϵ, thus satisfying the requirements in Definition 1. □

We now provide proof that X-Lock is reusable under Definition 2. Reusability means that the fuzzy extractor can support multiple independent enrollments of the same value, allowing users to reuse the same source in different contexts.

Theorem 4. *X-Lock is (ρ, σ)-reusable.*

Proof. The procedure *Gen* can be run multiple times on correlated strings of the same source, $S^1_{read}, \ldots, S^\rho_{read}$. Each time, *Gen* produces a different pair of values $(K_1, H_1), \ldots, (K_\rho, H_\rho)$. The security for each extracted string K_i should hold even in the presence of all the helper strings $H_1, \ldots, H_\rho$. In our specific case, $H = (V, (n, R, T))$. Having all H values does not compromise the security of the respective keys, as n, R are randomly determined and T is generated through hashing, making it pseudo-random by definition. Additionally, the vault V remains the same for each key, meaning that the overall reusability is only conditioned on the security of the vault V. We provide definition of the security of the pool vault in Definition 8. □

Notably, our model also provides insider security [5,6]. This means that the algorithm provides reusability even in the case where the attacker is given all the K_j for $j \neq i$. Each key K_j is the output of a hash function and provides no information about the input. Hence, a specific key cannot be used to infer information about another one, even in the presence of correlation in the input data between the two keys.

According to Definition 3, a fuzzy extractor is deemed robust if a user is able to detect any tampering with the public data H. We provide proof that X-Lock is indeed a robust algorithm.

Theorem 5. *X-Lock is a τ-robust fuzzy extractor with error δ.*

Proof. The attacker may attempt to change one or more in the helper data $H = (V, (n, R, T))$. Nevertheless, any alteration would be detected by the authentication token $T \leftarrow hash(K, R) = hash(hash(K_{pre}, n), R)$. Tampering with the vault V would lead to changes in the recovered values of K_{pre}, inevitably affecting the resulting hash. Furthermore, modifying the nonce n or the indexes R directly impacts the resulting hash. Additionally, changing token T to T' would require to find suitable values n', R', K'_{pre} such that $T' \leftarrow hash(hash(K'_{pre}, n'), R')$. This is infeasible, given the properties of cryptographic hash functions. Consequently, the pair (Gen, Rep) strictly adheres to the requirements specified in Definition 3, as any alterations introduced by an attacker can be effectively detected through the validation token T. □

5.2 Bias and Correlation Analysis

The scheme proposed in Algorithm 1 offers perfect secrecy when utilizing a perfectly random source F and employing uncorrelated bits to XOR with the bits from B. However, obtaining a perfectly random source using physical mediums poses considerable challenges due to issues related to bias and correlation between bits. Nevertheless, Algorithm 1 helps in mitigating these concerns. Notice that both bias and correlation are intrinsic properties of fuzzy sources. Some works in literature [23,24] propose methodologies to reduce or even remove dependencies between bits, but they either require high computational processing or extensive analysis of the physical medium. For instance, removing correlated bits requires a perfect identification of such relationships, which is a costly and partial procedure. Missing even one correlation could be significantly harmful. Instead, our protocol implicitly mitigates bias and correlation without making assumptions about their structure. The XOR-ation adopted in the locks inherently diminishes the final bias (see Sect. 3). The probability p_{cbias} of an XOR-ation outputting value 1 can be represented by the equation:

$$p_{cbias} = \sum_{i=0}^{\frac{|C|}{2}} \binom{|C|}{2i+1} \cdot p_{bias}^{2i+1}(1-p_{bias})^{|C|-2i-1} \tag{2}$$

The XOR-ation outputs 1 whenever there is an odd number of 1s in C. Thus, the summation between 0 and $|C|$ and the indexes $2i+1$ consider only the odd numbers. For instance, with $p_{bias} = 0.77$ and $|C| = 3$, $p_{cbias} \approx 0.579$. Consequently, an almost random source is derived from a skewed one. However, it is crucial to note that the same mechanism that reduces bias also amplifies errors. In Sect. 5.3, we provide an analysis of the cost in memory related to a particular error tolerance.

The XOR-ation also aids in mitigating the effects of correlations among bits in the source F. The correlation factor ϕ (see Sect. 3) only requires an estimate of the level of correlated bits without any specific assumption on its type. By employing it, we can evaluate how the correlation impacts the overall security of the linear system. Figure 3 illustrates the impact of correlation on bit security with varying numbers of elements in the XOR-ation, considering $|S_{pref}| = 2^{16}$, $|B| = 2^9$, $|L| = 33, |C| = 4$, and $\phi = 0.75$. We randomly drew the XOR-ation bits from the source F and measured the number of exposed bits from the random pool while bits from F were defined. The graph clearly illustrates the significant improvement achieved with an increasing number of elements in the XOR-ation. Starting with just a single element in the XOR-ation, the system fails to provide adequate security, as all the 2^9 pool bits become exposed after defining only a couple of bits. This is attributable to the relationships among locks in different bit-lockers. The setting allows one bit-locker to be entirely solved by merely defining a single bit from F, which, in turn, sets a value for all other variables in the bit-locker. This creates a cascading effect that subsequently unlocks multiple other bit-lockers, thus leading to the compromised security. However, by augmenting the number of elements in the XOR-ation to three, the

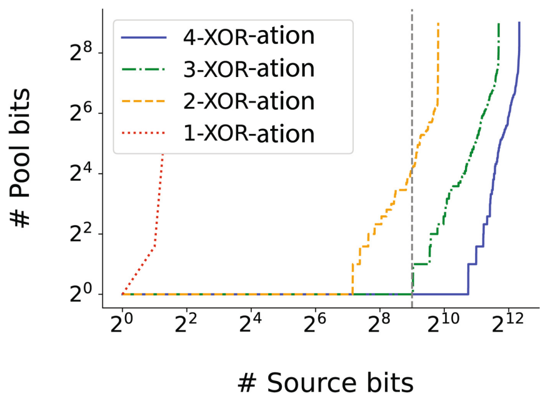

Fig. 3. Impact of correlation on bit security with varying number of elements in the XOR cascade. The x-axis considers the number of defined bits from the source. The y-axis shows the corresponding number of exposed bits from the random pool. Using a single element provides virtually no security. From three elements onwards the number of defined bits is greater than those in the protected pool (vertical, dashed line). (Color figure online)

number of required source bits becomes closely aligned with the actual number of protected bits from the pool (as denoted by the vertical dashed line). This indicates that a potential attacker would not gain any additional advantage by exploiting the correlation among variables, as the number of variables to be defined is equivalent to the number of bits in the random pool they are attempting to set. Furthermore, when employing four elements in the XOR-ation, the system significantly surpasses the reference line, demonstrating that the impact of correlation becomes negligible in this context. The higher the bias, the correlation and the security requirements to achieve adversarial non-advantageous contexts, the higher the number of bits in the XOR-ation. However, the higher $|C|$, the higher the resulting error rate in reconstructing the keys. We can estimate such value taking into account the absolute error rate e_{abs} of F.

5.3 Costs Analysis

Memory Cost. Procedure *Init* creates the supporting pool vault. The vault V contains $|B|$ bit-lockers, one for each bit in the random pool B. Each bit-locker hosts $|L|$ locks, with each lock being a single bit. To generate the locks, the algorithm has to choose a subset of bits from S_{pref}. If randomly chosen, the subset of indexes must be stored in order to permit subsequent recovery. However, this strategy would consume a lot more memory than V itself. The same applies to the storage of the set of indexes R related to each key. A more effective solution would be to design an index routine that can generate the corresponding bits on the fly. We deployed an example of a custom dynamic strategy that works as follows. To generate the requisite set of indices, we utilize a Pseudo Random

Number Generator (PRNG) and define a seed to ensure reproducibility. The PRNG generates a dependable sequence of random numbers with a given seed, though collisions may occur. To address collisions, we increase the colliding number until a fresh and distinct index is achieved. We apply modular arithmetic when the value reaches the upper limit. As previously discussed in Sect. 4, this routine is applicable to both $drawVaultIndexes$ and $drawKeyIndexes$. The dynamic generation of the subset of indexes does not entail any security implications when compared to the naive strategy of random generation and subsequent storage. In the latter case, the subset becomes publicly available, whereas in the former case, the algorithm is open-source as well. The system's security is not contingent on maintaining the secrecy of the indexes. The tradeoff between these two approaches is primarily related to performance. Dynamic index generation reduces memory costs, albeit with a manageable increase in computational overhead. Conversely, the naive strategy involves a straightforward lookup but necessitates storing each index, leading to higher memory expenses. In particular, to store all the indexes leveraging the static approach requires storing an index spanning between 1 and $|S_{pref}|$ for $|B|$ bit-lockers, each constituted by $|L|$ locks implementing a XOR-ation of $|C|$ bits for the vault and an index spanning between 1 and $|B|$ for $|K|$ bits for each key. Considering the dynamic approach, the cost for storing the pool vault indexes is rather nothing more than the bits required to represent the seed for the PRNG. The same holds for the set of indexes for each key. Let us denote the PRNG seed as $seed$ and its size in bits as $|seed|$. Hence, the cost for storing the vault boils down to $|B| \cdot |L| + |seed|$ bits using the dynamic approach from $|B| \cdot |L| \cdot (1 + |C| \cdot log_2(|S_{pref}|))$ bits obtained with the static approach. Procedure Gen generates a novel key K. It requires storing a nonce n, an authentication token T and the set of indexes R used to generate the key. The static approach requires $|n| + |T| + |K| \cdot log_2(|B|)$ bits, whilst the dynamic approach just necessitates of $|n|+|T|+|seed|$ bits. Procedure Rep does not require additional elements to be stored. Therefore, let us denote with $\mathcal{K}$ the total number of keys generated. Then, the total cost for the static strategy is $\mathcal{O}(|B| \cdot |L| \cdot (1 + |C| \cdot log_2(|S_{pref}|)) + \mathcal{K} \cdot (|n| + |T| + |K| \cdot log_2(|B|)))$ bits. Rather, leveraging the dynamic strategy decreases the total cost to $\mathcal{O}(|B| \cdot |L| + |seed| + \mathcal{K} \cdot (|n| + |T| + |seed|))$ bits.

Computational Cost. Procedure $Init$ necessitates the preferred state S_{pref} to generate the vault V. To determine the correct value, multiple strings from F are required to obtain a statistically relevant sample, often comprising hundreds of samples. The time to gather this sample may vary depending on the physical support used. For instance, SRAMs require a non-negligible amount of time (in the order of milliseconds) for the chip to discharge after shutdown [15]. As a result, collecting a sufficient sample might take several seconds. Once the preferred state is obtained, the procedure employs a series of XOR operations to compute the locks. Each lock utilizes $|C| - 1$ XORs for the XOR-ation and an additional XOR for the final bit. Assuming each XOR operation is $\mathcal{O}(1)$, the overall cost of computing all locks is $\mathcal{O}(|C| \cdot |B| \cdot |L|)$. Notably, procedure $Init$ is only computed once, and if energy is a limited resource, it can be performed before deploying

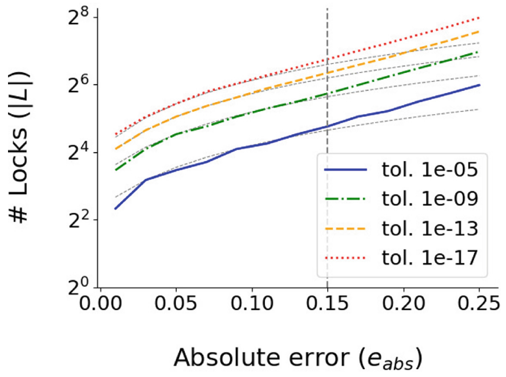

Fig. 4. Number of locks required to achieve a given error tolerance for varying absolute errors. The x-axis considers the absolute error. The y-axis uses logarithmic scale and considers the number of lockers. The gray dashed lines provide a linear reference for the four curves. The curves show an initial linear trend that slowly diverges to exponential at around value 0.85 for the error. (Color figure online)

the appliance, as the vault V can be made public. Procedure *Gen* utilizes the normal state S_{read} to recover the bits from the vault and generate a new key. It then draws a random nonce n and $|K|$ random values. Considering these operations to be $\mathcal{O}(1)$, the cost becomes $\mathcal{O}(|K|+1)$. Subsequently, it proceeds to recover the protected bit from each lock, requiring $|C|$ XORs for each unlocking operation. This process must be performed $|K| \cdot |L|$ times, resulting in a cost of $\mathcal{O}(|K| \cdot |C| \cdot |L|)$. Finally, it computes two hashes, denoted as $\mathcal{O}(2 \cdot t_{hash})$. Thus, the overall cost is $\mathcal{O}(|K| \cdot (1+|C| \cdot |L|)+1+2 \cdot t_{hash})$. Procedure *Rep* follows a similar pattern to *Gen*. For each lock, it unlocks the protected bit, incurring a cost of $\mathcal{O}(|K| \cdot |C| \cdot |L|)$. Additionally, it computes two hashes, yielding a total cost of $\mathcal{O}(|K| \cdot (1+|C|+|L|)+1+2 \cdot t_{hash})$. The dynamic index generation strategy adds extra computational cost to the algorithm. In particular, let us denote with t_{PRNG} the time required to query the PRNG and with *collisions* the number of collisions. Generating and reproducing the indexes for the vault takes $\mathcal{O}(t_{PRNG} \cdot |B| \cdot |L| \cdot |C| + collisions)$, whereas the same routine for the set of indexes for each key takes $\mathcal{O}(t_{PRNG} \cdot |K| + collisions)$. The decision to utilize dynamic index generation or static index storage depends on the specific scenario. Dynamic index generation proves advantageous in contexts characterized by limited memory resources and available energy supply. On the other hand, static index storage is a more viable option when energy is scarce but memory resources are more readily available.

Impact of Error Tolerance. In Sect. 5.1, we demonstrated that by adjusting the number of locks $|L|$ within a bit-locker, it is possible to manage the error tolerance e_{lock} of the system. In this analysis, we examine the impact of e_{lock} on $|L|$ and set four different levels of tolerance as reference points. Utilizing the formula provided in Sect. 5.1, we computed the number of expected locks required to achieve a given tolerance for each specified error rate. The results are plot in Fig. 4.

The four lines in the graph represent distinct error tolerances. We observe that the trend is approximately linear for lower rates of absolute errors. However, at around an error rate of 0.85, the trends start to diverge and become exponential. Notably, the curve corresponding to $e_{lock} = 1e - 05$ presents an interesting exception. Here, the number of locks exhibits an under-linear trend that extends until an error rate of 0.8. This graph effectively demonstrates that our solution experiences linear growth for low to medium error rates, resulting in linear memory and computational costs within that range.

6 Implementation and Comparison

In order to further validate the correctness and efficiency of our approach, we implemented the algorithm on a physical test-bed using a 2.3 GHz Intel Core i5 quad-core processor with 8 GB of 2133 MHz LPDDR3 memory. We employed a SRAM as our PUF, which was easily accessible using a Keystudio MEGA 2560 R3 microcontroller board with an 8 KB SRAM. The selection of an SRAM PUF was motivated by several factors, including its user-friendly nature, ease of development, cost-effectiveness, and its widespread availability as a piece of hardware. To determine the preferred state S_{pref} of the SRAM, we collected 200 strings and computed the majority value for each bit. We found that 88% of the bits presented a stable value, with the remaining 12% showing an average absolute error of 0.0223 and a maximum error of 0.0455.

The algorithm was implemented in C language and compared against both Woo et al. [28], which is the best-performing algorithm in terms of both memory and computational complexity, and Canetti et al. [5,6], which is the algorithm providing the highest security guarantees. We evaluated three sets of parameters based on security levels: 80-bit, 128-bit, and 256-bit. For each level, we performed two experiments with different setups. Table 3 presents the parameters used for each security level.

The values for $|C|$ were carefully chosen in order to mitigate the effect of correlation as shown in Sect. 5.2. We did not consider the bias in this simulation because also the comparison works did not consider it. We decided to experiment with two specific values for the correlation factor: $\phi = 0.75$ represents a realistic correlation factor for SRAMs [19], and $\phi = 0.00$ corresponds to a limit case where all the PUF bits are independent. In such scenario, no correlation relationship affects the bits of the PUF meaning that no XOR-ation is mandatory, net of the bias phenomenon. The correlation factor directly impacts the error tolerance by determining the number of available equations for the linear system (see Sect. 5.1). The sixth column (*i.e.,* Exp. key error rate) reports the expected error rates in recovering the key for each configuration. We kept it at around 1 error

Table 3. Parameters for the three security levels.

Chosen params			PUF params		Resulting params		
$\|K\|$	$\|L\|$	$\|C\|$	ϕ	e_{abs}	e_{lock}	$e_{bitlock}$	Exp. key error rate
80	64	2	0.75	0.15	0.255	$2.243 \cdot 10^{-5}$	0.0018
80	256	2	0.00	0.25	0.375	$3.089 \cdot 10^{-5}$	0.0025
128	64	2	0.75	0.15	0.255	$2.243 \cdot 10^{-5}$	0.0029
128	256	2	0.00	0.245	0.370	$1.483 \cdot 10^{-5}$	0.0019
256	64	3	0.75	0.10	0.244	$8.519 \cdot 10^{-6}$	0.0022
256	256	3	0.00	0.175	0.363	$4.891 \cdot 10^{-6}$	0.0013

every 500. We set $|S_{read}| = 2^{16}$ based on the size of our SRAM and $|B| = 2^8$ for the random pool size. Expected key error rate parameter is resulting from the choice of the other parameters. In particular, recall Eq. 1 which is the probability that a bit-locker is wrongly unlocked. Then, the probability that all bit-lockers of a given key are successfully unlocked is $(1 - e_{lock})^{|K|}$. Finally, the expected key error rate is the probability that at least one bit-locker of a given key is wrongly unlocked and it is given by $1 - (1 - e_{lock})^{|K|}$.

We measured the time needed to perform the *Rep* procedure. We performed 100000 experiments. To perform a fair comparison with the state-of-the-art, we downloaded the code of Woo et al. [28] from the public repository[1], we implemented by ourselves the algorithm proposed by Canetti et al. [5,6], and run them on the same appliance in order to obtain results originating from the same evaluation environment. In fact, differences affecting computational power among distinct machines could compromise the reliability of our experiments.

Table 4 summarizes the results and compares our work with [28] and [5,6]. Different e_{abs} values are arising from other parameters, such as key length $|K|$, correlation factor ϕ and expected key error rate. In particular, we set each e_{abs} value as the maximum we could afford in each configuration before performance degradation and security flaws significantly increased.

We considered two state-of-the-art constructions for the comparison with our work. Our choice was driven by the guarantees they offer and the performances, in terms of memory requirements and computational time, they achieve. In particular we chose the proposal of Canetti et al. [5,6] and the proposal of Woo et al. [28]. The former one provides optimal security guarantees, offering reusability, insider security and robustness, while also addressing source correlation. The latter one offers reusability and robustness with an easily manageable memory overhead and linear error tolerance.

The other works either do not provide a sufficient level of security, exhibiting data leakage or not supplying reusability and/or robustness, or are completely unfeasible. In particular:

- Secure sketch-based constructions [2,25,26] suffer from data leakage;

[1] https://github.com/KU-Cryptographic-Protocol-Lab/Fuzzy_Extractor.

Table 4. Performance comparisons. The time value includes both the generation and the reproduction procedure.

Solution	$\|K\|$	e_{abs}	Memory	Time
Canetti et al. [5,6] ($\phi = 0.00$)	80	0.25	$\mathcal{K} \cdot 143.95$ MB	43.50 s
	128	0.245	$\mathcal{K} \cdot 216.20$ MB	57.71 s
	256	0.175	$\mathcal{K} \cdot 431.02$ MB	104.99 s
Woo et al. [28] ($\phi = 0.00$)	80	0.30	$\mathcal{K} \cdot 297$ B	4.62 s
	128	0.20	$\mathcal{K} \cdot 559$ B	15.51 s
	256	0.11	$\mathcal{K} \cdot 1087$ B	52.209 s
Our work ($\phi = 0.75$)	80	0.15	2064 B $+ \mathcal{K} \cdot 64$ B	0.054 ms
	128	0.15	2064 B $+ \mathcal{K} \cdot 64$ B	0.086 ms
	256	0.10	2064 B $+ \mathcal{K} \cdot 64$ B	0.222 ms
Our work ($\phi = 0.00$)	80	0.25	8208 B $+ \mathcal{K} \cdot 64$ B	0.205 ms
	128	0.245	8208 B $+ \mathcal{K} \cdot 64$ B	0.305 ms
	256	0.175	8208 B $+ \mathcal{K} \cdot 64$ B	0.845 ms

– The construction proposed by Cheon et al. [9] improves the proposal of Canetti et al. [5,6] by reducing the memory requirements. Nevertheless, the price to pay for such improvement is the introduction of a significant amount of hashing operations, which are costly, determining an increase of the already expensive computational cost;
– Fuller et al. [13] and Apon et al. [1] constructions lack of reusability.

The memory cost of Woo et al. [28] algorithms is comparable with our implementation incurring an initial overhead for storing the vault V. However, our memory consumption grows slower with the number of generated keys $\mathcal{K}$. For instance, the memory cost for a security class of 128-bit and $\phi = 0.75$ would become equivalent to [28] after generating 5 keys. In terms of computation time, our algorithm outperforms [28] by at least three orders of magnitude. The significant difference is due to the use of more efficient operations, mainly XORs and a limited number of hashes, in our work. On the other hand, [28] employs more resource-intensive LWE-based cryptography and error correction codes.

As for the error tolerance, our implementation with $\phi = 0.75$ handles smaller amounts of error, which is a side effect of the XOR-ation mechanism (see Sect. 5.1). The correlation factor ϕ and the resulting number of available bits in the SRAM PUF significantly affect the overall performances. In fact, by setting $\phi = 0.00$ the error tolerance of our solution becomes superior to [28] in two security levels, while being almost comparable in the third level. This setting is equivalent of having a source F with 2^{17} bits and a correlation factor $\phi = 0.75$.

To validate the result from Sect. 5.1, we also measured the number of errors committed in each security class. We measured 163 errors for class 80-bits, 259 for class 128-bits, and 129 for class 256-bits. These are in line with the expected errors calculated in Table 3 (respectively 180, 280, and 220) We then checked the claim made in Sect. 5.1 by assessing the capability of an adversary to combine different locks for gaining additional knowledge. We performed 100000 cycles, with every cycle randomly shuffling the locks and adding them together. We monitored the total number of elements forming the XOR-ation. Figure 5 plots the minimum number of elements observed in a XOR-ation by combining multiple locks. The graph clearly shows a growing trend that finds its upper bound at a value that is half of the total dimension of the source. This result validates the claim made in Sect. 5.1 on the security of the pool vault. The vast majority of locks shares no common elements, hence combining them greatly increase the XOR-ation dimension. The presence of locks with more than half common elements has little to no impact on the overall security.

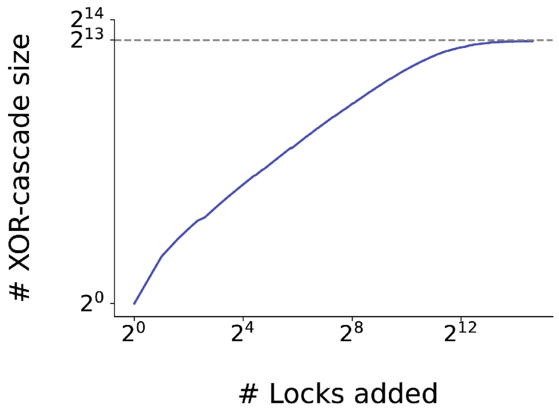

Fig. 5. XOR-ation size while combining multiple locks. The x-axis considers the number of locks added. The y-axis considers the number of elements forming the XOR-ation. Both axes are logarithmic. The graph considers the worst case, that is the minimum number of elements in the resulting cascade. (Color figure online)

7 Conclusion

The proliferation of resource-constrained devices within the IoT domain has caused considerable challenges concerning privacy and security. Traditional methods of storing digital keys in non-volatile memory have proven intricate and costly, prompting the need for alternative solutions. This paper introduces X-Lock, a novel computational fuzzy extractor specifically designed to address the limitations faced by traditional solutions in resource-constrained IoT devices. X-Lock employs a unique approach, utilizing the preferred state of a noisy source to encrypt a random string of bits, which subsequently serves as a seed to generate multiple secret keys. This design not only ensures both reusability (even insider security), and robustness, but also effectively mitigates bias and correlation, enhancing overall security. To substantiate the claims of X-Lock, a comprehensive theoretical analysis is presented, encompassing security considerations and detailed implementation insights. The rigorous analysis validates the effectiveness and security of the proposed model. To evaluate the superiority of X-Lock, an extensive set of practical experiments is also conducted, and the results are compared against existing approaches. The experimental findings demonstrate the efficacy of our proposed model, showcasing its comparable memory cost (approximately 2.4 KB for storing 5 keys of 128 bits) and remarkable speed gains, which outperform the state-of-the-art solution by three orders of magnitude (0.086 ms compared to 15.51 s). By offering reusability, insider security, and robustness, X-Lock presents a compelling solution to the challenges posed by traditional key storage methods. The comprehensive theoretical analysis and practical experiments affirm the superior performance of X-Lock, making it a promising advancement for enhancing privacy and security in the rapidly evolving IoT landscape.

Acknowledgments. This work was partially supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU.

References

1. Apon, D., Cho, C., Eldefrawy, K., Katz, J.: Efficient, Reusable fuzzy extractors from LWE. In: Dolev, S., Lodha, S. (eds.) CSCML 2017. LNCS, vol. 10332, pp. 1–18. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60080-2_1
2. Boyen, X.: Reusable cryptographic fuzzy extractors. In: Proceedings of the 11th ACM Conference on Computer and Communications Security, pp. 82–91. Association for Computing Machinery (2004). https://doi.org/10.1145/1030083.1030096
3. Boyen, X., Dodis, Y., Katz, J., Ostrovsky, R., Smith, A.: Secure remote authentication using biometric data. In: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22–26, 2005. Proceedings 24, pp. 147–163 (2005)
4. Canetti, R., Dakdouk, R.R.: Obfuscating point functions with multibit output. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 489–508. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78967-3_28
5. Canetti, R., Fuller, B., Paneth, O., Reyzin, L., Smith, A.: Reusable fuzzy extractors for low-entropy distributions. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology - EUROCRYPT 2016, pp. 117–146. Springer, Berlin Heidelberg, Berlin, Heidelberg (2016)
6. Canetti, R., Fuller, B., Paneth, O., Reyzin, L., Smith, A.: Reusable fuzzy extractors for low-entropy distributions. J. Cryptol. **34**, 1–33 (2021)
7. Chang, C.H., Zheng, Y., Zhang, L.: A retrospective and a look forward: fifteen years of physical unclonable function advancement. IEEE Circuits Syst. Mag. **17**(3), 32–62 (2017)
8. Chen, B., Ignatenko, T., Willems, F., Maes, R., van der Sluis, E., Selimis, G.: High-rate error correction schemes for sram-pufs based on polar codes. arXiv preprint arXiv:1701.07320 (2017)
9. Cheon, J.H., Jeong, J., Kim, D., Lee, J.: A reusable fuzzy extractor with practical storage size: Modifying canetti et al'.s construction. In: Information Security and Privacy: 23rd Australasian Conference, ACISP 2018, Wollongong, NSW, Australia, July 11–13, 2018, Proceedings 23, pp. 28–44 (2018)

10. Dodis, Y., Ostrovsky, R., Reyzin, L., Smith, A.: Fuzzy extractors: how to generate strong keys from biometrics and other noisy data. SIAM J. Comput. **38**(1), 97–139 (2008). https://doi.org/10.1137/060651380
11. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Trans. Inf. Theory **29**(2), 198–208 (1983)
12. Fuller, B., Meng, X., Reyzin, L.: Computational fuzzy extractors. In: Advances in Cryptology-ASIACRYPT 2013: 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1–5, 2013, Proceedings, Part I 19. pp. 174–193. Springer (2013)
13. Fuller, B., Meng, X., Reyzin, L.: Computational fuzzy extractors. Inf. Comput. **275**, 104602 (2020)
14. Hiller, M.: Key derivation with physical unclonable functions. Ph.D. thesis, Technische Universität München (2016)
15. Liu, M., Zhou, C., Tang, Q., Parhi, K.K., Kim, C.H.: A data remanence based approach to generate 100% stable keys from an sram physical unclonable function. In: 2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), pp. 1–6 (2017)
16. Lynn, B., Prabhakaran, M., Sahai, A.: Positive results and techniques for obfuscation. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 20–39. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_2
17. Maes, R., van der Leest, V., van der Sluis, E., Willems, F.: Secure key generation from biased PUFs. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. 517–534. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48324-4_26
18. Obermaier, J., Immler, V., Hiller, M., Sigl, G.: A measurement system for capacitive puf-based security enclosures. In: Proceedings of the 55th Annual Design Automation Conference, pp. 1–6 (2018)
19. Rahman, M.T., Hosey, A., Guo, Z., Carroll, J., Forte, D., Tehranipoor, M.: Systematic correlation and cell neighborhood analysis of sram puf for robust and unique key generation. J. Hardw. Syst. Secur. **1**, 137–155 (2017)
20. Roel, M.: Physically unclonable functions: Constructions, properties and applications, pp. 148–160. Katholieke Universiteit Leuven, Belgium pp (2012)
21. Seo, M., Kim, S., Lee, D.H., Park, J.H.: Emblem:(r) lwe-based key encapsulation with a new multi-bit encoding method. Int. J. Inf. Secur. **19**, 383–399 (2020)
22. Shafarevich, I.R., Remizov, A.O.: Linear algebra and geometry. Springer Science & Business Media (2012)
23. Suzuki, M., Ueno, R., Homma, N., Aoki, T.: Efficient fuzzy extractors based on ternary debiasing method for biased physically unclonable functions. IEEE Trans. Circuits Syst. I Regul. Pap. **66**(2), 616–629 (2018)
24. Ueno, R., Suzuki, M., Homma, N.: Tackling biased pufs through biased masking: A debiasing method for efficient fuzzy extractor. IEEE Trans. Comput. **68**(7), 1091–1104 (2019)
25. Wen, Y., Liu, S.: Reusable fuzzy extractor from lwe. In: Information Security and Privacy: 23rd Australasian Conference, ACISP 2018, Wollongong, NSW, Australia, July 11–13, 2018, Proceedings, pp. 13–27 (2018)
26. Wen, Y., Liu, S.: Robustly reusable fuzzy extractor from standard assumptions. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11274, pp. 459–489. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03332-3_17

27. Wilde, F., Gammel, B.M., Pehl, M.: Spatial correlation analysis on physical unclonable functions. IEEE Trans. Inf. Forensics Secur. **13**(6), 1468–1480 (2018). https://doi.org/10.1109/TIFS.2018.2791341
28. Woo, J., Kim, J., Park, J.H.: Robust and reusable fuzzy extractors from non-uniform learning with errors problem. Comput. Mater. Continua **74**(1) (2023)