Securing Systems with Scarce Entropy: LWE-Based Lossless Computational Fuzzy Extractor for the IoT

Christopher Huth[1], Daniela Becker[2], Jorge Guajardo[2], Paul Duplys[1], and Tim Güneysu[3]

[1] Robert Bosch GmbH, Stuttgart, Germany
[2] Robert Bosch LLC, Pittsburgh, USA
[3] University of Bremen & DFKI, Bremen, Germany

Abstract. With the advent of the Internet of Things, lightweight devices necessitate secure and cost-efficient key storage. Since traditional secure storage is expensive, the valuable entropy could originate from noisy sources, for which fuzzy extractors allow strong key derivation. While providing information-theoretic security, fuzzy extractors require large amount of input entropy to account for entropy loss in the key extraction process. It has been shown by Fuller *et al.* [20] that the entropy loss can be reduced if the requirement is relaxed to computational security based on the hardness of the Learning with Errors problem. Using this computational fuzzy extractor, we show how to construct a device-server authentication system providing outsider chosen perturbation security and pre-application robustness. We present the first implementation of a *lossless* computational fuzzy extractor where the entropy of the source equals the entropy of the key on a constrained device. The implementation needs only 1.45KB of SRAM and 9.8KB of Flash memory on an 8-bit microcontroller. We compare our implementation to existing work in terms of security, while achieving no entropy loss.

Keywords: Computational fuzzy extractor · Learning with errors · Authentication system · Implementation

1 Introduction

After file sharing, e-commerce, and social media, the next generation of the Internet—the Internet of Things (IoT)—is connecting machines to machines. These IoT devices range from sensors and security cameras to vehicles, production machines, buildings and smart cities. It is expected that there will be 50 billion connected IoT devices by 2020 [19]. Due to connectivity, security is a major concern for IoT systems.

1.1 Security in the IoT and Problem Description

The IoT will consist of countless devices with a connection to the Internet and constrained in terms of memory, power supply and computational power. Sensor nodes, in particular, change the way we used to think about computer-based systems. They will become the eyes and ears for our everyday ubiquitous computing world. That way, cyber-physical systems can directly influence our physical environment with their collected data, e.g. a fire door unlocks upon smoke detection or a car brakes due to input from various sensors detecting an obstacle. Therefore, it is widely recognized that authenticity of their data and communication will be a requirement to guarantee the safe and secure running of IoT systems.

Cryptographic mechanisms ensuring secure deployment and operation of IoT devices rely on high-entropy keys. Harvesting entropy on IoT devices, however, turns out to be a challenging task because of the inherently constrained nature of these devices. The number and type of peripherals available on such devices is kept at a minimum for cost reasons. Moreover, since noise-free high-entropy sources are not generally available, IoT devices must rely on noisy entropy sources [4]. Examples of noisy entropy sources[4] include biometrics [12], quantum information [7] and physically unclonable functions (PUF) [24,40,21,48]. Especially PUFs are an emerging trend in IoT systems and can be found in devices ranging from small chip card microcontrollers like NXP's SmartMX2 to modern high-performance FPGAs and MPSoCs like Xilinx UltraScale family. Due to the lightweight, resource-constrained nature of IoT systems, securely storing keys on IoT devices remains a challenge. Here, PUFs are considered an attractive solution as the key can be seen to be embedded intrinsically in hardware.

As a result, potential entropy sources and, in turn, the overall amount of available entropy are scarce. Efficient use of the available entropy is therefore a necessary prerequisite for building secure and privacy-respecting IoT systems.

1.2 Background and Aim

As mentioned previously, PUFs are attractive because they provide for secure and cheap cryptographic key storage, even in constrained environments as those found in IoT systems. To deal with the PUF noise and, more generally, be able to derive secure cryptographic keys from noisy sources, fuzzy extractors were introduced [17]. Fuzzy extractors are essentially comprised of two procedures. The generate procedure establishes a key and a helper data from a measurement. The reproduce procedure takes a noisy measurement, alongside with the helper data, and reproduces the exact same key. It can be formally shown that the public helper data leaks negligible (or no) information about the derived key [17,10,20]. Because of this, fuzzy extractors can be found in cryptographic protocols [10,23,3,13]. Security of fuzzy extractors constructions has been shown in the information-theoretic sense. However, it comes at the cost of an entropy loss equal to the difference between the measurement (source) entropy and the entropy of the extracted key. In constrained devices, such entropy loss results in increased costs for the overall system (more entropy from a PUF, implies additional SRAM cells, oscillators, etc. which, in turn result in addtional area and therefore increased cost).

[4] Clearly, not all such sources are meant to be used with constrained IoT devices.

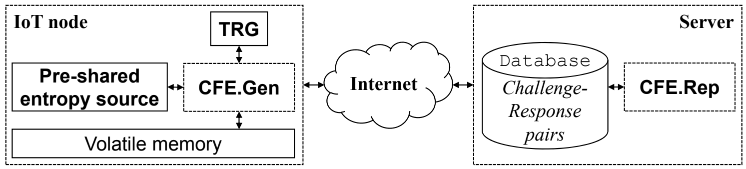

Fig. 1: Figure of our computational fuzzy extractor in an IoT setting.

In [20], Fuller *et al.* introduced a *computational* fuzzy extractor (CFE), which can be transformed into a *lossless* fuzzy extractor, i.e., an extractor that has *no* entropy loss. In contrast to information theoretic constructions where the extracted key is derived from the noisy source measurement, the main idea behind a computational fuzzy extractor is to use the measurement to *encrypt* a uniformly chosen secret. Because the encryption and decryption relies on the Learning with Errors (LWE) problem [42], recovering the secret is still possible with a noisy measurement that is sufficiently close to the initial one. Such a construction has the additional advantage of being post-quantum secure, thanks to inherited security from LWE.

In this paper, our aim is to design, implement and evaluate the feasibility of a cheap IoT node and corresponding system supporting lightweight mutual authentication in a post-quantum world. We achieve this via algorithms which minimize the required (software or hardware) resources and the entropy required for key derivation (and therefore for authentication). Post quantum security follows directly from our use of computational fuzzy extractors whose construction is based on the LWE problem. Our evaluation indicates that *it is* feasible to implement such (CFE) schemes in highly constrained environments. In the process of showing this feasibility results, we provide novel constructions of random number generators for ultra-constrained environments and algorithms which minimize the area required for the CFE implementation.

1.3 System Model

We describe how our proposed system can be implemented as in Fig. 1. An IoT node is connected with a server over the Internet. The node itself is equipped with a true random number generator (TRG), a memory, a CPU which runs the generate procedure (CFE.Gen) and a pre-shared secret entropy source. Here, we assume a TRG which requires no additional hardware costs and we show in this paper the validity of this assumption. Further we assume a strong PUF as pre-shared secret entropy source in our system and we note that the implementation of the PUF is outside the scope of this paper. The TRG provides freshness for each protocol run and the volatile memory stores intermediate values during calculation.

The server, on the other hand, has access to a database which holds the corresponding challenge-response pairs for each legitimate PUF which are pre-measured during an enrollment phase in a secure environment. Also, the server runs the reproduce logic of our computational fuzzy extractor ($\mathsf{CFE.Rep}$). We assume the server—or the cloud—to be protected with high-security measures. We claim our proposed mutual authentication system offers outsider chosen perturbation security and pre-application robustness.

1.4 Adversarial Model

In vision of the IoT depicted in Fig. 1, we assume a strong adversary. The adversary can control the communication channel between a node and a server at will, due to the wireless nature of the IoT. In addition, the adversary has limited access to the node itself. In particular, we assume that the adversary cannot perform extensive physical attacks or invasive or semi-invasive side-channels attacks (which require unlimited access to the device for extended periods of time) but he has the ability to read out secrets from standard non-volatile memory and change them (replace them). As it is standard, we assume that the PUF in the system possesses a tamper evidence property, so that if the adversary tries to learn the secret stored in the PUF, the PUF behavior will change significantly or be destroyed. Finally, we assume that all the security functionality (algorithms) related to the CFE is implemented in such a way that it cannot be modified but is well-known to the adversary (as it is standard in cryptography).

1.5 Contribution

In this paper, we investigate the feasibility of a lossless CFE for typical IoT devices. To show the limits, we choose a very constrained 8-bit device, as well as a 32-bit device for comparison. The latter speeds up the client's generate procedure from 34.9 to 0.4 seconds. We explore a system based on the lossless CFE construction in terms of efficiency and complexity. We summarize our contributions as follows:

- **CFE system.** We show how a computational fuzzy extractor can be included securely in a client-server authentication system by taking advantage of reverse and robust fuzzy extractors. Our construction provides outsider chosen perturbation security and pre-application robustness. Additionally, our construction immediately achieves post-quantum security due to its theoretical relation to the LWE problem.
- **Client-side implementation of a reverse and robust CFE on constrained devices.** To our knowledge, we present the first actual implementation[5] of a *lossless* computational fuzzy extractor [20] on resource-constrained devices (an 8-bit AVR microcontroller with 2.5KB RAM and a 32-bit ARM Cortex-M3 microcontroller with 96 KBytes of RAM).

[5] The work of [26] describes the implementation of a Trapdoor CFE. Our focus is on a *plain* CFE, where the additional confidence information of a Trapdoor CFE is not available. In contrast, we have implemented a *lossless* CFE, based on LWE instead of LPN, and on a constrained device rather than a normal computer.

– **True random number generation.** We propose a new construction for generating random numbers from a physical noise source available on off-the-shelf microprocessors. The proposed construction achieves uniform randomness required in our CFE system.
– **Parameter setting and optimized implementation of a lossless CFE.** We show how to optimize the algorithms given by Fuller *et al.* [20] in a way that reduces their memory footprint. This enables the application of lossless CFE in the embedded domain where the devices generally have a limited amount of memory. In addition, we show how to determine suitable parameters and discuss the impact of the parameters' choice on the implementation size and performance. Finally, we compare our setting with related work for 80-bit, 128-bit and 256-bit security.

1.6 Outline

The outline of this paper is as follows: In Section 2, we introduce notation, necessary background and previous constructions. We then present our reverse and robust computational fuzzy extractor in Section 3. In Section 4 we show how to use our previous system construction for an authentication protocol. We give implementation details and optimizations for the used algorithms in Section 5. In Section 6 we discuss parameters for a lossless implementation and present resulting memory requirement. We evaluate our implementation in Section 7 in terms of memory and performance. A thorough security analysis for our system construction and our protocol, as well as a comparison with related work is conducted in Section 8. At last, we discuss possible pre-shared secret entropy sources for our system in Section 9. We conclude this paper in Section 10.

2 Preliminaries

We follow the same notation as in [20]. For a random variable $X = X_1||\dots||X_n$, where each X_i is over some alphabet $\mathcal{Z}$, we denote by $X_{1,\dots,k} = X_1||\dots||X_k$. We write for a distinguisher D (or a class of distinguishers $\mathcal{D}$) the *computational distance* between X and Y as $\delta^D(X,Y) = |\mathbb{E}[D(X)] - \mathbb{E}[D(Y)]|$. We denote by $\mathcal{D}_{s_{sec}}$ the class of randomized circuits which output a single bit and have size at most s_{sec}. U_n denotes the uniformly distributed random variable on $\{0,1\}^n$. We denote random variables by capitalized letters, e.g. X, matrices or vectors by bold letters, e.g. $\mathbf{A}$ or $\mathbf{x}$, and elements in a vector or samples from a random variable by lowercase letters, e.g. x. We will write $\mathcal{M}$ to denote a metric space with an associated distance function dis. We will denote the finite field with q elements by $\mathbb{F}_q$ and the corresponding vector space of dimension m over $\mathbb{F}_q$ by $\mathbb{F}_q^m$. We denote the binary logarithm with log. We denote an efficient algorithm as probabilistic polynomial time (PPT). We use a Truly Random Number Generator (TRNG) to derive truly random binary sequences. Furthermore, we use Message Authentication Codes (MAC), where MAC uses secret key x and message m as inputs and outputs $\sigma = \mathsf{MAC}_x(m)$. A MAC is verified with function

Ver on inputs x, m and σ and outputs if the MAC is valid (*yes*) or not (*no*). For our constructions we use an encoding function Enc which uses a key $\mathbf{x}$, a matrix $\mathbf{A}$ and a string $\mathbf{w} \in \mathcal{M}$. The corresponding decoding function Dec takes a vector $\mathbf{b}$ and matrix $\mathbf{A}$ as inputs and outputs key $\mathbf{x} = \mathsf{Dec}(\mathbf{b}+\mathbf{e}, \mathbf{A})$ if $\mathbf{b} = \mathsf{Enc}_{\mathbf{x}}(\mathbf{A}, \mathbf{w})$ and where $\mathbf{e}$ is some small noise vector.

2.1 Learning with Errors

Regev introduced the LWE problem [42,43] as a generalization of the Learning Parity with Noise (LPN) problem. We recall the decisional version of the problem.

Definition 1 (Decisional LWE [42]) *Let n be a security parameter. Let $m = m(n) = \mathtt{poly}(n)$ be an integer and $q = q(n) = \mathtt{poly}(n)$ be a prime. Let $\mathbf{A}$ be uniformly distributed over $\mathbb{F}_q^{m\times n}$, X be uniformly distributed over $\mathbb{F}_q^n$ and χ be an arbitrary distribution on $\mathbb{F}_q^m$. The decisional version of the LWE problem, denoted $\mathsf{dist\text{-}LWE}_{(n,m,q,\chi)}$, is to distinguish the distribution $(\mathbf{A}, \mathbf{A}X + \chi)$ from the uniform distribution over $(\mathbb{F}_q^{m\times n}, \mathbb{F}_q^m)$.*

We say that $\mathsf{dist\text{-}LWE}_{(n,m,q,\chi)}$ is (ϵ, s_{sec})-secure if no (probabilistic) distinguisher of size s_{sec} can distinguish the LWE instances from uniform except with probability ϵ.

2.2 Computational Fuzzy Extractors

Formalized in [17], fuzzy extractors (FE) can be used to derive keys from noisy measurements. They have been proven secure in the information-theoretical sense. A FE consists of two procedures—Gen and Rep. Whereas Gen "generates" public helper data from a measurement, Rep tries to "reproduce" a shared secret from a noisy measurement and the helper data.

Fuller *et al.* [20] show how to build computational fuzzy extractors (CFE) to derive longer keys compared to information-theoretical secure fuzzy extractors when input entropy remains the same at the cost of achieving only computational security. Their construction is based on the LWE problem. By using the variant of LWE [18], which uses a uniformly random distribution (rather than a discretized Gaussian) and choosing suitable parameters, Fuller *et al.* also show how to construct a *lossless* CFE, i.e., a CFE exhibiting no entropy loss. We first recall the definition of a CFE [20].

Definition 2 (Computational Fuzzy Extractor [20, Definition 2.5]) *Let $\mathcal{W}$ be a family of probability distributions over $\mathcal{M}$. A pair of randomized procedures "generate" (Gen) and "reproduce" (Rep) is a $(\mathcal{M}, \mathcal{W}, \ell, t)$-computational fuzzy extractor that is (ϵ, s_{sec})-hard with error δ if Gen and Rep satisfy the following properties:*

- *The generate procedure Gen on input $w \in \mathcal{M}$ outputs an extracted string $R \in \{0,1\}^\ell$ and a helper string $P \in \{0,1\}^*$.*

– *The reproduction procedure* Rep *takes an element* $w' \in \mathcal{M}$ *and a bit string* $P \in \{0,1\}^*$ *as inputs. The correctness property guarantees that if* $\mathsf{dis}(w, w') \leq t$ *and* $(R, P) \leftarrow \mathsf{Gen}(w)$*, then* $\Pr[\mathsf{Rep}(w', P) = R] \geq 1 - \delta$*, where the probability is over the randomness of the procedures* (Gen, Rep)*. If* $\mathsf{dis}(w, w') > t$ *then no guarantee is provided about the output of* Rep*.*
– *The security property guarantees that for any distribution* $W \in \mathcal{W}$*, the string* R *is pseudorandom conditioned on* P*, that is* $\delta^{\mathcal{D}_{sec}}((R, P), (U_\ell, P)) \leq \epsilon$*.*

2.3 *Lossless* Computational Fuzzy Extractor

As mentioned previously, [20] observes that by careful choice of parameters and using the LWE version introduced in [18], one can achieve a lossless CFE as shown in Construction 1. Intuitively, the Gen procedure takes $\mathbf{w} \leftarrow W$, where W is a uniform distribution over $\mathbb{F}_{\rho q}^m$, as input and outputs a key r and the helper data p. The secret vector $\mathbf{x} \in \mathbb{F}_q^n$ is chosen uniformly, but only the first k blocks of $\mathbf{x}$ result in the key r. This follows directly from the ability to extract pseudorandom bits, which, in turn, follows from [[1], Theorem 3] that proves that $\mathbf{x}$ has simultaneously many hardcore bits.

The Rep outputs the key r for a given noisy $\mathbf{w}'$ and helper data p, if the error t is not too big. Basic operations in Gen and Rep are matrix-vector multiplications and vector-vector additions, or subtractions, in a prime field. The overall efficiency of this construction relies on the function Decode, which we describe in Section 5.2.

Construction 1 ([20]) *Let* n *be a security parameter and let* $m \geq 3n$ *and* $k = n/2$*. Let* q *be a prime. Define* Gen, Rep *as follows:*

Gen	**Rep**
1. ***Input:*** $w \leftarrow W$ *(where* W *is some distribution over* $\mathbb{F}_{\rho q}^m$*)*	1. ***Input:*** (w', p) *(where* $\mathsf{dis}(w, w') \leq t$*)*
2. *Sample* $\mathbf{A} \in \mathbb{F}_q^{m \times n}, \mathbf{x} \in \mathbb{F}_q^n$ *uniformly*	2. *Parse* p *as* $(\mathbf{A}, \mathbf{c})$*; let* $\mathbf{b} = \mathbf{c} - w'$
3. *Compute* $p = (\mathbf{A}, \mathbf{A}\mathbf{x} + w)$, $r = \mathbf{x}_{1,\ldots,k}$	3. *Let* $\mathbf{x} = \mathsf{Decode}_t(\mathbf{A}, \mathbf{b})$
4. ***Output:*** (r, p)	4. ***Output:*** $r = \mathbf{x}_{1,\ldots,k}$

For Construction 1 to be lossless, the parameters in the construction need to satisfy several conditions. The results from Döttling and Müller-Quade [18] enable [20] to split a sampled measurement $\mathbf{w}$ into blocks w_i. Each block then represents a coordinate in the vector $\mathbf{w}$, where m is the number of coordinates in $\mathbf{w}$ and each w_i has a bit width of ρq. The key r gets derived from k hardcore samples x_i from $\mathbf{x}$, so the total number of bits in r is $k \log q$. Fuller *et al.* observe that for a lossless construction we need to satisfy $m \log \rho q = k \log q$, meaning the measurement vector $\mathbf{w}$ has a higher dimension, but each element has less bits, than the key r. The above discussion is *visualized* in (1), where we are abusing the log notation to illustrate the number of bits needed to represent the operands. The formal statement is contained in Theorem 1. Theorem 2 summarizes the security of the lossless CFE construction and it is proven in [20].

$$\log \mathbf{w} = \log \overbrace{\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \\ \vdots \\ w_m \end{pmatrix}}^{\log \rho q} = m \log \rho q = k \log q = \log \overbrace{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{pmatrix}}^{\log q} = \log \mathbf{x} \tag{1}$$

Theorem 1 ([18, Theorem 6]) *Let n be a security parameter and let $\sigma \in (0,1)$ be an arbitrarily small constant. Let $q = q(n)$ be a prime and $m = m(n) =$ $\mathsf{poly}(n)$ be a integer with $m \geq 3n$. Let $\rho = \rho(n) \in (0, 1/10)$ be such that $\rho q \geq 2n^{1/2+\sigma}m$. If there exists a PPT-algorithm that solves $\mathsf{dist\text{-}LWE}_{(n,m,q,\mathcal{U}([-\rho q, \rho q]))}$ with non-negligible probability, then there exists an efficient quantum-algorithm that approximates the decision-version of the shortest vector problem (GAPSVP) and the shortest independent vectors problem (SIVP) to within $\tilde{O}(n^{1+\sigma}m/\rho)$ in the worst case.*

Theorem 2 ([20, Theorem 4.7]) *Let n be a security parameter and let the number of errors $t = c \log n$ for some positive constant c. Let d be a positive constant (giving us a trade-off between running time of Rep and $|w|$). Consider the Hamming metric over the alphabet $\mathcal{Z} = [-2^{b-1}, 2^{b-1}]$, where $b = \log 2(c/d + 2)n^2 = O(\log n)$. Let W be uniform over $\mathcal{M} = \mathcal{Z}^m$, where $m = (c/d + 2)n = O(n)$. If GAPSVP and SIVP are hard to approximate within polynomial factors using quantum algorithms, then there is a setting of $q = \mathtt{poly}(n)$ such that for any polynomial $s_{sec} = \mathtt{poly}(n)$ there exists $\epsilon = \mathtt{ngl}(n)$ such that the following holds: Construction 1 is a $(\mathcal{M}, W, m \log |\mathcal{Z}|, t)$-computational fuzzy extractor that is (ϵ, s_{sec})-hard with error $\delta = e^{-\Omega(n)}$). The generate procedure Gen takes $O(n^2)$ operations over $\mathbb{F}_q$, and the reproduce procedure Rep takes expected time $O(n^{4d+3})$ operations over $\mathbb{F}_q$.*

Remark 1 *We have chosen to include the formal theorems because they allow us to discuss the parameters in a precise manner. This is in contrast to a more informal exposition of the parameters, which can result in confusion when discussing the parameter selection in the next sections.*

Construction 1 is defined by its parameters. As motivated before, it is possible to achieve a lossless setting. In this section we state the parameters, their relation and constraints.

- $|W|$: The length of the source.
- $|X_{1,\dots,k}|$: The length of the key.

– t: Number of errors that can be supported.
– n: LWE security parameter (number of field elements in X).
– q: The size of the field.
– ρ: The fraction of the field needed for error sampling.
– m: The size of each number of samples in the LWE instance.
– k: The number of hardcore bits in X. ($k \log q = m \log \rho q$)

The goal is to minimize the entropy loss, meaning $|W| - |X_{1,...,k}|$ should be as small as possible. In the best case this would yield $|W| = |X_{1,...,k}|$, translating to $m \log(\rho q) = k \log q$. For security we need n to be greater than some minimum n_0 and $q = \texttt{poly}(n)$. Also, we need a bound on the error t so that efficient decoding is still possible. The larger we choose the dimension m, the more samples our decoder can extract, so the more errors we can support. Substituting, the error t depends on m, which we chose minimal. The previous discussion results in the following collection of constraints. A detailed derivation is given in [20]:

$$n_0 < n - k \tag{2}$$
$$m = 3n \tag{3}$$
$$q = \texttt{poly}(n) \tag{4}$$
$$\rho q = 2n^{\frac{1}{2}+\sigma} m \tag{5}$$
$$m \log(\rho q) = k \log q \tag{6}$$

The parameter t, i.e., the level of noise our construction can tolerate, is further explored in Section 7.2.

2.4 Reverse and Robust Fuzzy Extractor

In the following we adapt the definition of a *reverse fuzzy extractor* from [49] to our construction, since our Rep-procedure does not reproduce the original response w but computes the extracted string r that is generated by Gen.

Definition 3 (Reverse Fuzzy Extractor [49, Definition 1]) *A pair of PPT algorithms* (Gen, Rep) *is a* $(\mathcal{M}, m, m', t)$*-reverse fuzzy extractor if it has the following two properties for correctness and security, respectively:*

– *If* $r \leftarrow (r, p) = \mathsf{Gen}(w)$ *and* $\mathsf{dis}(w, w') \leq t$*, then w.h.p* $\mathsf{Rep}(w', p) = r$
– *A PPT adversary* $\mathcal{A}$ *with input* $p \leftarrow (r, p) = \mathsf{Gen}(\mathsf{w})$ *outputs* w *with probability negligible in* m'

where $w' \in \mathcal{M}$ *and* w *is sampled according to distribution* W *over* $\mathcal{M}$ *with min-entropy* m*.*

Security against outsider chosen perturbation attacks describes a stronger notion of security for reverse fuzzy extractors.

Definition 4 (Outsider Chosen Perturbation Security [49]) *An* $(\mathcal{M}, m, m', t)$*-reverse fuzzy extractor as defined in Definition 3 is secure against outsider chosen perturbation attacks if there is no PPT adversary* $\mathcal{A}$ *that wins the following security game with more than negligible advantage in* m'*:*

– $\mathcal{A}$ *chooses a distribution* $W \in \mathcal{M}$ *with min-entropy* m
– *Challenger* $\mathcal{C}_{PS}$ *randomly chooses* $w \xleftarrow{\$} W$
– $\mathcal{A}$ *adaptively chooses* $e_i \in \mathcal{M}$, *s.t.* $\forall i : |e_i| \leq t$ *and invokes the oracle* Gen
– Gen *computes* $(r_i, p_i) = \mathsf{Gen}(w_i = w + e_i)$ *and outputs* p_i *to* $\mathcal{A}$
– $\mathcal{A}$ *outputs a guess* w^* *to* $\mathcal{C}_{PS}$
– $\mathcal{A}$ *wins if* $w^* = w$

Finally, we restate the definition of a robust fuzzy extractor by Dodis *et al.* [16].

Definition 5 (Robust Fuzzy Extractor [16, Definition 6]) *A* (m, l, t, ϵ)-*fuzzy extractor that is* (ϵ, s_{sec})-*hard has pre-application robustness* δ *if there is no PPT adversary that given* p *from* $(r, p) = \mathsf{Gen}(w)$ *outputs a* p' *s.t.* $p' \neq p$ *and* $\mathsf{Rep}(w', p') \neq \perp$ *where* $\mathsf{dis}(w, w') \leq t$ *with probability higher than* δ, $|r| = l$ *and* w *has min-entropy* m.

Note that specific to our construction we do not consider post-application robustness as defined in [16, Definition 6], where the adversary is provided with r from $(r, p) = \mathsf{Gen}(w')$ since in our case r corresponds to a secret value that is not communicated over a (possibly insecure) channel.

3 Reverse and Robust CFE

We use the reverse fuzzy extractor mechanism of van Herrewege *et al.* [49] for authentication and enhance it to a robust fuzzy extractor secure against outsider chosen perturbation attacks.

The reverse fuzzy extractor effectively flips the Gen and Rep procedures for a prover and a verifier, here device and server respectively. The server transmits a challenge c to the device holding a pre-shared entropy source, e.g. a PUF. The device stimulates the source and gets a noisy response $\mathbf{w}'$. Then a helper data p is generated for this noisy response $\mathbf{w}'$ and p is sent back to the server. The server knows a $\mathbf{w}$ close to $\mathbf{w}'$ for the given challenge c from a previous enrollment phase. The server can correct his $\mathbf{w}$ with the helper data p to retrieve $\mathbf{w}'$. That way the resource-constrained device does not have to do the computationally intense decoding of a linear random code. A reverse computational fuzzy extractor is depicted in Fig. 2, which further implments our system from Fig. 1.

Since the manipulation of the public helper data p poses an attack vector [14], we enhance our reverse computational fuzzy extractor to a reverse and robust computational fuzzy extractor. We do this by integrating a MAC to prove the helper data p was generated by the device, i.e., $\sigma = \mathsf{MAC}_{\mathbf{x}_{1,\ldots,n}}(\mathbf{A}, \mathbf{A}\mathbf{x} + \mathbf{w}')$. This enhancement is depicted in Fig. 3. Our construction can detect tampered helper data $\tilde{p}$, with $p \neq \tilde{p}$. Possible outputs of our Rep construction are therefore extended with a failure symbol $\perp$, when the verification function Ver fails.

Information-theoretic secure fuzzy extractors can leak entropy with a code-offset or syndrome construction [17]. The leakage between the generated helper data and secret key can be prevented by adapting techniques from [34], i.e.,

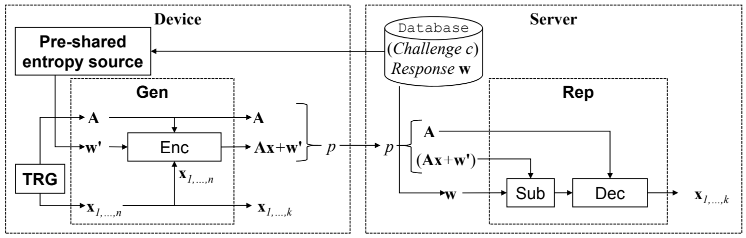

Fig. 2: Figure of our reverse computational fuzzy extractor authentication scheme.

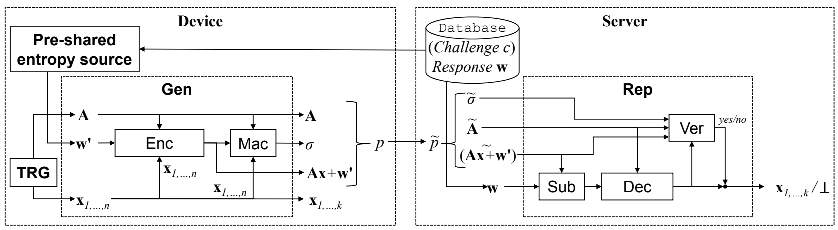

Fig. 3: Figure of our reverse and robust computational fuzzy extractor authentication scheme.

debiasing. Another solution is presented in [44] where a small noise is added to the response to still be correctable. This is where Construction 1 benefits from the LWE problem. If the measurement $\mathbf{w}$ is biased, then we can also add a small (pseudo)random noise for debiasing and the definition of a $(\mathcal{M}, \mathcal{W}, \ell, t)$-computational fuzzy extractor still holds.

The server knows a challenge-response pair $(c, \mathbf{w})$. The device outputs the helper data p, which consists of matrix $\mathbf{A}$, the encoded $\mathbf{Ax}+\mathbf{w}' = \mathsf{Enc}_{\mathbf{x}_{1,\dots,n}}(\mathbf{A}, \mathbf{w}')$ and a MAC $\sigma = \mathsf{Mac}_{\mathbf{x}_{1,\dots,n}}(\mathbf{A}, \mathbf{Ax} + \mathbf{w}')$ of the previous two. The server then subtracts $\mathbf{b} = \mathbf{Ax} + \mathbf{w}' - \mathbf{w} = \mathsf{Sub}(\mathbf{Ax} + \mathbf{w}', \mathbf{w})$ and decodes $\mathbf{b}$ to retrieve $\mathbf{x}_{1,\dots,n} = \mathsf{Dec}(\mathbf{b}, \mathbf{A})$. With this retrieved $\mathbf{x}_{1,\dots,n}$ and the public matrix $\mathbf{A}$, the server can verify if the helper data was changed. For verification the server calculates $\mathbf{w}' = \tilde{\mathbf{A}} \cdot \mathbf{x}_{1,\dots,n} - (\mathbf{Ax} \tilde{+} \mathbf{w}')$, where $\mathbf{w}'$ should be close to $\mathbf{w}$. If the verification is successful the first k hardcore bits of $\mathbf{x}_{1,\dots,n}$ are recovered as a shared secret $\mathbf{x}_{1,\dots,k}$.

Security proofs of our construction being a reverse and a robust computational fuzzy extractor, secure against outsider chosen perturbation attacks, are given in Section 8.1.

4 Mutual Authentication Protocol

In this section, we demonstrate how to actually use the authentication scheme from Fig. 3. Delvaux *et al.* reviewed several PUF-based lightweight entity authentication protocols [13]. For our authentication scheme we chose the reverse

FE protocol [49] as a basis, working with strong PUFs. A modified version of the reverse FE protocol also offers mutual authentication, but uses weak PUFs [33]. Also, the generate and reproduce procedures are the ones from our reverse and robust computational fuzzy extractor as depicted in Fig. 3. Our system model and assumptions are the same as in Section 1, but are formally described in the following.

4.1 System Model

Our scheme consists of at least three parties, namely an issuer $\mathcal{I}$, a device $\mathcal{D}$ and a server $\mathcal{S}$. The adversary is denoted with $\mathcal{A}$. $\mathcal{S}$ can access a database `DB`, where all IDs of legitimate devices $\mathcal{D}$ alongside their pre-measured challenge-response pairs are listed. $\mathcal{I}$ initializes and maintains `DB`.

4.2 Trust Model and Assumptions

Issuer $\mathcal{I}$ and server $\mathcal{S}$ We assume $\mathcal{I}$ and $\mathcal{S}$ to be trusted, which is a typical assumption in PUF-based authentication protocols. $\mathcal{D}$, $\mathcal{S}$ and `DB` are initialized in a secure environment.

Device $\mathcal{D}$ We assume $\mathcal{D}$ to be a passive device, meaning it cannot start a communication. Also, we assume a present strong PUF, i.e. the pre-shared entropy source in Fig. 3 is implemented as a strong PUF, denoted as $f_i(\cdot)$ with i stating the PUF's uniqueness. Furthermore, we assume a TRNG, a hash function and the generate procedure of Fig. 3 on $\mathcal{D}$.

Adversary $\mathcal{A}$ We assume an active adversary who has full control over the communication channel between $\mathcal{D}$ and $\mathcal{S}$, i.e., $\mathcal{A}$ can eavesdrop, modify and intercept all protocol messages and can send arbitrary messages to $\mathcal{S}$ and $\mathcal{D}$ alike. The goal of the adversary is to impersonate either the server or the device. We allow $\mathcal{A}$ to know whether an authentication was successful or not. Additionally, $\mathcal{A}$ can read any information stored in non-volatile memory before and after protocol execution, e.g. during an unsecure distribution chain. However, $\mathcal{A}$ cannot get responses of the entropy source $f(\cdot)$ and cannot access temporary data of $\mathcal{D}$, e.g. intermediate results, while the protocol is executed[6].

4.3 Protocol

The system is initialized by issuer $\mathcal{I}$, who stores a random identifier ID in the non-volatile memory of device $\mathcal{D}$. Also, $\mathcal{I}$ creates $r > 0$ challenge-response pairs $(c_1, \mathbf{w}_1), \ldots, (c_r, \mathbf{w}_r)$ during a secure enrollment phase from $f(\cdot)$ of device $\mathcal{D}$. Challenge-response pairs are stored in database `DB` with the corresponding ID.

[6] We note that an adversary could use side-channel attacks to extract these intermediate values and one should harden a system with side-channels aware designs, which are outside the scope of this paper.

Device $\mathcal{D}$		**Server** $\mathcal{S}$
$f_i(\cdot), ID$		$\texttt{DB} = (ID', (c_1, \mathbf{w}_1), \ldots, (c_r, \mathbf{w}_r))$
	$\xleftarrow{\texttt{auth}}$	
	$\xrightarrow{ID}$	**if** $(ID \neq ID')$ **then** $\perp$
		$j \xleftarrow{\mathsf{U}} \{1, \ldots, r\}$
$\mathbf{w}'_j \xleftarrow{\mathsf{R}} f_i(c_j)$	$\xleftarrow{c_j, \eta}$	$\eta \xleftarrow{\mathsf{U}} \{0, 1\}^l$
$(\mathbf{x}_{1,\ldots,k}, p_j) := \mathsf{Gen}(\mathbf{w}'_j)$		
$a := \mathsf{Hash}(ID, \eta, \mathbf{x}_{1,\ldots,k}, p_j)$	$\xrightarrow{p_j, a}$	
		$\mathbf{x}_{1,\ldots,k} := \mathsf{Rep}(\mathbf{w}_j, p_j)$
		if $(\mathsf{Hash}(ID, \eta, \mathbf{x}_{1,\ldots,k}, p_j) \neq a)$ **then** $\perp$
if $(\mathsf{Hash}(a, \mathbf{x}_{1,\ldots,k}) \neq b)$ **then** $\perp$	$\xleftarrow{b}$	$b := \mathsf{Hash}(a, \mathbf{x}_{1,\ldots,k})$

Fig. 4: Mutual authentication protocol

For the actual authentication protocol, as shown in Fig. 4, the server $\mathcal{S}$ starts by sending an authentication request `auth` to device $\mathcal{D}$. $\mathcal{D}$ answers with its identifier ID and if ID is not present in `DB` then the protocol aborts. Next, $\mathcal{S}$ selects a random challenge-response pair $(c_j, \mathbf{w}_j)$ from `DB` and a random nonce η and transmits (c_j, η) to $\mathcal{D}$. Upon reception, $\mathcal{D}$ stimulates the pre-shared entropy source $\mathbf{w}'_j \xleftarrow{\mathsf{R}} f_i(c_j)$, generates $(\mathbf{x}_{1,\ldots,k}, p_j) := \mathsf{Gen}(\mathbf{w}'_j)$ with our reverse and robust computational fuzzy extractor, calculates hash $a := \mathsf{Hash}(ID, \eta, \mathbf{x}_{1,\ldots,k}, p_j)$ and sends (p_j, a) to $\mathcal{S}$. In return, $\mathcal{S}$ reproduces the secret $\mathbf{x}_{1,\ldots,k}$ with $\mathbf{x}_{1,\ldots,k} := \mathsf{Rep}(\mathbf{w}_j, p_j)$ using the premeasured $\mathbf{w}_j$ from `DB`. $\mathcal{S}$ checks if $\mathsf{Hash}(ID, \eta, \mathbf{x}_{1,\ldots,k}, p_j) = a$ and aborts if not. Otherwise, $\mathcal{S}$ computes $b := \mathsf{Hash}(a, \mathbf{x}_{1,\ldots,k})$ and sends b to $\mathcal{D}$ which accepts if $\mathsf{Hash}(a, \mathbf{x}_{1,\ldots,k}) = b$ and aborts otherwise.

We show in Section 8.2 that our protocol holds correctness and mutual authentication.

5 Optimizations and Details for Used Algorithms

We implemented the reverse and robust computational fuzzy extractor-based authentication scheme shown in Fig. 3. As the client, we use a device with an 8-bit ATmega32u4 microprocessor, the server implementation runs on a 3.2GHz single core machine with 8GB RAM.

5.1 Generate Procedure

For the Gen procedure, as defined in Construction 1, notice that there are three main variables, namely $\mathbf{A}$, $\mathbf{x}$ and $\mathbf{w}$. These three variables are drawn from a

Algorithm 1 Gen

(Input $w \in \mathbb{F}_{\rho q}^{m}$)
Sample $\mathbf{x} \in \mathbb{F}_{q}^{n}$
for $i = 1 \dots m$ **do**
    $acc = 0$
    for $j = 1 \dots n$ **do**
        Sample $a_{ij} \leftarrow$ TRG
        Output a_{ij}
        $acc = a_{ij} \times x_j + acc$
    end for
    Sample $w_i \leftarrow$ Pre-shared entropy source
    $b_i = acc + w_i$ // $\mathbf{Ax} + \mathbf{w} = b$
    Output b_i
end for
Output $r = \mathbf{x}_{1,\dots,n/2}$

predefined distribution, i.e., $\mathbf{w} \leftarrow W$ (where W is some distribution over $\mathbb{F}_{\rho q}^{m}$), $\mathbf{A} \in \mathbb{F}_{q}^{m \times n}$ uniformly and $\mathbf{x} \in \mathbb{F}_{q}^{n}$ uniformly.

The size of matrix $\mathbf{A}$ is a bottleneck. We solve this by not storing $\mathbf{A}$ in memory, but rather computing it "on the fly" as every element $a_{ij} \in \mathbf{A}$ is used only once in the Gen procedure. The elements a_{ij} are the outputs of a true random number generator TRG, described in Section 5.1. We also apply the same idea for the measurement $\mathbf{w}$, so that the complete vector $\mathbf{w}$ does not need to be held in memory, but rather gets measured element by element. The pseudocode for our encoding is given in Algorithm 1. With this algorithm the memory overhead gets reduced as $\mathbf{A}$ gets streamed out of the Gen procedure. The public helper data p is still $p = (\mathbf{A}, \mathbf{Ax} + \mathbf{w})$ and also the output of the secret $r = \mathbf{x}_{1,\dots,n/2}$. So, our improvement meets the definition of Construction 1. The memory footprint can be further reduced by shifting the sampling of $\mathbf{x}$ in the outer loop. That way only the upper half of $\mathbf{x}$ needs to be stored, as it is the output secret r. The lower half of $\mathbf{x}$ is not reused and can be stored in a temporary variable.

Multiplication Our implementation requires a fast multiplication of two 30-bit values, i.e., $a_{ij} \times x_j$ in Algorithm 1. For this, we adopt the column-wise multiplication from Gura *et al.* [25]. We represent each number by an array of four bytes and the multiplication result by an array of eight bytes. This leaves the two, or four respectively, most significant bits without meaning. The idea is shown in Fig. 5.

Modular Reduction Solinas describes a way to efficiently calculate a result b of reducing a modulo p, where p is a prime and a is less than p^2 [46]. We choose our 30-bit prime $p = 2^{30} - 2^{18} - 1 = 1073479679$ and every 60-bit integer a can

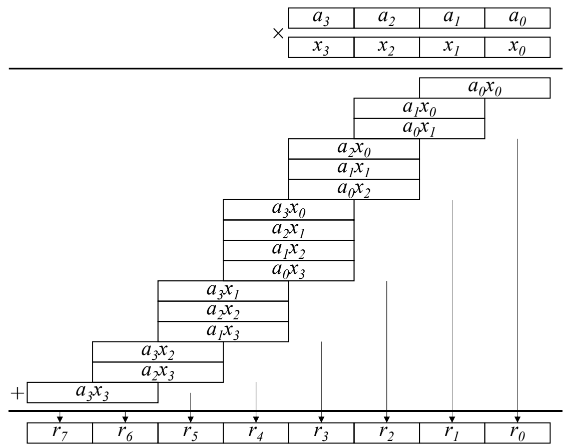

Fig. 5: Figure of our implemented multi-precision multiplication from [25]. The operation shows $r = a \times x$, where a and x are 30-bit numbers and r is a 60-bit number.

be written as

$$a = a_9 \cdot 2^{54} + a_8 \cdot 2^{48} + a_7 \cdot 2^{42} + a_6 \cdot 2^{36} + a_5 \cdot 2^{30} + a_4 \cdot 2^{24} + a_3 \cdot 2^{18} + a_2 \cdot 2^{12} + a_1 \cdot 2^{6} + a_0$$

where each a_i is a 6-bit integer. As a concatenation of 6-bit words, this can be denoted by $a = (a_9||a_8||\ldots||a_0)$. The expression for b is then

$$b := T + S_1 + S_2 + S_3 + S_4 \mod p$$

where the 30-bit terms are given by

$$\begin{aligned} T &= (\ a_4 \ || \ a_3 \ || \ a_2 \ || \ a_1 \ || \ a_0\ ) \\ S_1 &= (\ a_6 \ || \ a_5 \ || \ 0 \ || \ a_6 \ || \ a_5\ ) \\ S_2 &= (\ 0 \ || \ a_7 \ || \ a_7 \ || \ 0 \ || \ a_7\ ) \\ S_3 &= (\ a_8 \ || \ a_8 \ || \ 0 \ || \ a_8 \ || \ 0\ ) \\ S_4 &= (\ a_9 \ || \ a_9 \ || \ a_9 \ || \ 0 \ || \ a_9\ ) \end{aligned}$$

We apply this modular reduction after every multiplication and addition in Algorithm 1.

True Random Number Generation Kristinsson explored in his work [30] the feasibility of using the noise delivered by one unconnected analog pin to an internal voltage comparator as a TRG. He also showed the presence of a strong bias, when using a single pin for measurement. This bias depends mostly on the environmental temperature.

We overcome this bias by measuring two analog pins and comparing them. Thus, we cancel out any biasing effects from the environment. The idea is depicted in Fig. 6 with the two unconnected analog input pins—Analog0 and Analog1. As noise source we utilize the random atmospheric noise, which occurs

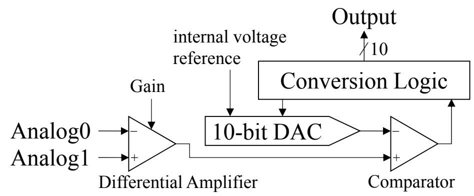

Fig. 6: Figure of the internal analog comparator.

during the analog to digital conversion. We first amplify the unconnected analog pins by a gain of 200×, meaning that we further amplify the noise. Second, the signal is converted with a 10-bit digital-to-analog converter (DAC), which works with a 2.56V internal voltage reference.

As a post-processing step, we take the least significant bit out of each 10-bit output value to generate a stream of random bits. We further estimate the min-entropy of our random bit sequence with the tests described in [5], yielding an estimated min-entropy of 3.95 bits per byte. We input at least twice the estimated min-entropy into the HMAC-SHA-256 as privacy amplification in order to guarantee a nearly full entropy random number [5]. All in all, we need to sample at least 122 bits from our noise source to generate a 30-bit random number. Our random numbers pass the tests of the NIST Statistical Test Suite [39].

If the microcontroller lacks an analog comparator, the entropy source for a TRG can also be, e.g., the SRAM [27] with the construction proposed in [3].

5.2 Decode

The Construction 1 is only efficient if the function Decode_t is efficient. Fuller *et al.* [20] presented a simple decoding algorithm given in Construction 2 that can correct $O(\log n)$ errors in polynomial time using the given random linear code.

Construction 2 ([20, Construction 4.5]) *We consider a setting of* (n, m, q, χ) *where* $m \geq 3n$. *We describe* Decode_t*:*

1. *Input* $\mathbf{A}$, $\mathbf{b} = \mathbf{A}\mathbf{x} + \mathbf{w} - \mathbf{w}'$.
2. *Randomly select rows without replacement* $i_1, \ldots, i_{2n} \leftarrow [1, m]$.
3. *Restrict rows from* $\mathbf{A}, \mathbf{b}$ *to rows* $i_1, \ldots, i_{2n}$*; denote these* $\mathbf{A}_{i_1,\ldots,i_{2n}}, \mathbf{b}_{i_1,\ldots,i_{2n}}$.
4. *Find* n *linearly independent rows in* $\mathbf{A}_{i_1,\ldots,i_{2n}}$. *If no such rows exist, output* $\perp$ *and stop.*
5. *Denote* $\mathbf{A}', \mathbf{b}'$ *as these* n *restrictions of* $\mathbf{A}_{i_1,\ldots,i_{2n}}, \mathbf{b}_{i_1,\ldots,i_{2n}}$ *(respectively) to these rows. Compute* $\mathbf{x}' = (\mathbf{A}')^{-1}\mathbf{b}'$.
6. *If* $\mathbf{b} - \mathbf{A}\mathbf{x}'$ *has more than* t *nonzero coordinates, go to step 2.*
7. *Output* $\mathbf{x}'$.

As the authors of Construction 2 remark, their construction has not been optimized for constants. Also, the presented algorithm, when used in the computationally setting of Construction 1 will always output a correct key R, meaning Construction 2 will always have $\mathbf{x}' = \mathbf{x}$ as an output or it aborts when an

Algorithm 2 Decode

1: Input $\mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{x} + \mathbf{w} - \mathbf{w}'$, *limit*
2: **while** $limit > 0$ **do**
3: $\quad \mathbf{A}' = n$ rows of $\mathbf{A}$ randomly selected
4: $\quad$ **if** $\mathbf{A}'$ has full rank **then**
5: $\quad\quad \mathbf{b}' =$ same n restrictions of $\mathbf{b}$ as in $\mathbf{A}'$
6: $\quad\quad \mathbf{x}' = (\mathbf{A}')^{-1}\mathbf{b}'$
7: $\quad\quad$ **if** $\mathbf{b} - \mathbf{A}\mathbf{x}'$ has $\leq t$ nonzero coordinates **then**
8: $\quad\quad\quad$ Output $\mathbf{x}'$
9: $\quad\quad$ **end if**
10: $\quad$ **end if**
11: $\quad limit = limit - 1$
12: **end while**
13: Output $\perp$

non-invertible $\mathbf{A}'$ was selected. Note that Construction 2 does not necessary terminate when the error is too big or the row restrictions in step 5 are chosen badly, i.e., the same n linearly independent rows in $\mathbf{A}$ are selected, so Construction 2 could run infinitely. To avoid this issues, we optimized the decoding algorithm such that the behavior is defined for the case $\mathsf{dis}(\mathbf{w}, \mathbf{w}') > t$. We specifically optimized the following points:

- The row selection and restriction in steps 2, 3 and 5 can be replaced by a deterministic selection, so that rows will not be selected twice and that the algorithm terminates when all possibilities have been tried. This avoids the infinite loop problem of Construction 2.
- The algorithm could end too early in step 4 when disadvantageous rows were selected, e.g., in the first selection round, even if the error $t = 0$. To avoid this and to make our algorithm more robust, we introduce a parameter *limit* to try a minimum amount of selected row combinations.

Algorithm 2 describes our Decode function. The input *limit* is the maximum number of decoding attempts to find a solution and thereby avoids an infinite loop. If the algorithm outputs a $\perp$ then the error t was too big with high probability. We implemented the inversion of $\mathbf{A}'$ via Gauss-Jordan elimination, which runs in $O(n^3)$. Note that all operations are calculated in $\mathbb{F}_q$. Our Decode function returns a valid solution, if it finds n noise-free elements in $\mathbf{b}$. If the error t gets large, say $t > m/2$, and a solution has to be found, then the randomly selected restrictions can be replaced by a deterministic selection. The number of all possible restrictions is given by $\binom{m}{n}$. In an IoT setting this number of possible restrictions can still be feasible, since the decoding happens on a server.

6 Lossless Implementation Parameters

Regev [42] and Peikert [41] show that the $\mathsf{dist\text{-}LWE}_{(n,m,q,\chi)}$, as defined in Definition 1, is secure when the distribution χ is Gaussian. This holds with a discretized Gaussian distribution Ψ_α with variance $(\alpha q)^2/(2\pi)$.

If we change the error source from a discretized Gaussian distribution to a uniform one, we are bounded by Theorem 1. There, the problem is reduced to $\mathsf{dist\text{-}LWE}_{(n,m,q,\mathcal{U}([-\rho q,\rho q]))}$. Note that the error ranges from $[-\rho q, \rho q]$ and no longer from $[-q, q]$. Since this uniform error distribution is the noise level α, it should be high enough to hold security. Since we can not influence α directly, we have to increase ρ to increase the noise level. By Theorem 1 [18] we know the following bound on $2\sqrt{n} \leq \alpha q \leq \frac{\rho q}{mn^\sigma}$. It is clear, in order to maximize α, we have

$$\alpha = \frac{\rho}{mn^\sigma}, \tag{7}$$

where ρ has the most influence on the noise level. In order to still have a lossless construction we substitute this bound with the constraints above, clarifying that the noise level is given by the dividing factor ρ.

$$\alpha = \frac{\rho}{n^\sigma 3n} \tag{8}$$

The constraint $m \log(\rho q) = k \log q$, with $k = n/2$ and $m = 3n$, of the lossless construction dictates this ρ:

$$\log(\rho q) = \frac{1}{6} \log q, \tag{9}$$

meaning that by increasing the sample size $m = 3n$ we need the bit width of one element in w to be 1/6 of the bit width of one element in $\mathbf{x}$, so that we still can extract $k = n/2$ hardcore blocks as key. This is, again, visualized in (1).

For a *lossless* construction we choose parameters as described in Section 2.3. We found that parameters $n = 256, m = 768, k = 128, q = 1073479679$ yield $\log(q) \approx 30$ and $\log(\rho q) \approx 5$, thus making the construction lossless with $m \log(\rho q) - k \log(q) = 768 \times 5 - 128 \times 30 = 0$. This results in a minimum security parameter $n_0 \approx 128$. The parameters match with the ones chosen by Lindner and Peikert [31]. Their parameters of $n = 256, m = 768, q = 4093$ achieve a security level of 128 bit. From an implementation perspective our 30-bit prime fits well into 4 byte machine words, as do the elements of the key, each with a bit width of 30. The bit width of our source $\mathbf{w}$ is 5 for each element, which also is well suited as it fits into a single byte. There is room for further optimization so that elements of $\mathbf{w}$ fill a byte completely. However, this would also increase the dimension n and m to maintain the lossless construction, thus making decoding harder. We found our parameters are a good trade-off between security parameter n_0, unused bits in the byte architecture and the decoding time.

We conduct a security analysis in Section 8.3.

7 Evaluation

Our system, as shown in Fig. 3, requires $m \log(\rho q) = 3840$ bits from the fuzzy source and the lossless CFE extracts $k \log(q) = 3840$ bits as key. The key has full entropy, since the secret $\mathbf{x}$ is chosen uniformly. For the lossless construction, the fuzzy measurement $\mathbf{w}$ is assumed to come from an uniform distribution. The helper data p is matrix $\mathbf{A}$, vector $\mathbf{Ax} + \mathbf{w}'$ and the MAC σ, summing up to a total of 5,921,536 bits (ca. 740KB).

7.1 Implementation Cost on the Device

We implemented our system, as described in Sections 3 and 5, on the low-cost Atmel 8-bit AVR RISC-based microcontroller with 32KB self-programming flash program memory, 2.5KB SRAM and 1KB EEPROM.

For an efficient encoding, which is a matrix-vector multiplication and a vector-vector addition, the size of the parameter n is important. As we chose $\log(\rho q) = 5$, $\log(q) = 30$, $n = 256$ and $m = 768$, then $\mathbf{w}$ consists of $\log(\rho q) \times m =$ 3840 bits, $\mathbf{x}$ consists of $\log(q) \times n = 7680$ bits and $\mathbf{A}$ needs $\log(q) \times n \times m =$ 5898240 bits. Whereas, $\mathbf{w}$ and $\mathbf{x}$ is still feasible on a microcontroller in terms of memory usage, matrix $\mathbf{A}$ does not scale. With the optimizations from Section 5 we only need to store $\mathbf{x}$ in memory during the Gen procedure.

Additionally, we need some functionality on the microcontroller, e.g., a serial connection, mathematical operations and generation of a MAC. As MAC we use a HMAC-SHA-256. Our own files require ca. 6KB and the auxilirary files need ca. 17.8KB of the SRAM. A detailed memory footprint of our implementation per module is given in Table 2 in Appendix A. Table 3 in Appendix A sums up the overall memory footprint.

One run of the generation procedure, i.e., sampling of the source, encoding and construction of the MAC takes 34.9 seconds on average. A detailed timing profile is given in Table 4 in Appendix B. If all random numbers would be sampled via a TRG, the generation time would be infeasible, so we use these as a seed for a pseudorandom function (PRG). We also implemented the generation procedure on a 32-bit ARM Cortex-M3, which performs a Gen in 441ms, due to the internal 32-bit architecture and an overall higher clock speed.

7.2 Performance on the Server

We implemented the other part of our system, the server, as described in Construction 1 and 2. We also implemented our improvements for Algorithm 2. For this we used NTL[7] for fast matrix operations on finite fields.

We can correct $t = O(\log n)$ errors, as given in Theorem 2, and are bounded by the same limitations as Fuller *et al.* [20] for decoding in polynomial time. Recall that an error refers to a word in source $\mathbf{w}$, so a single or multiple bit flips in one element of vector $\mathbf{w}$ is one error. In Fig. 7 one can see that the

[7] http://www.shoup.net/ntl/

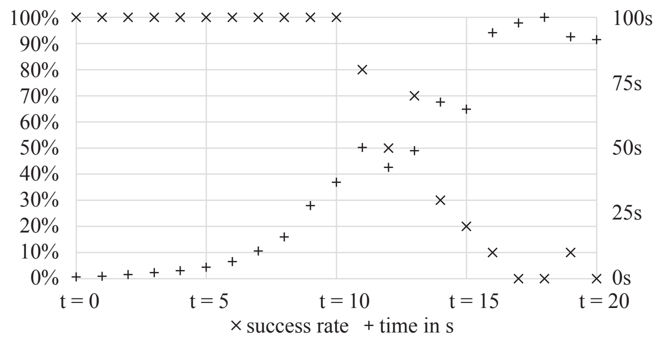

Fig. 7: Figure of $\mathsf{Rep}(\mathsf{Gen}(w))$ simulation with $n = 256, m = 3n$ and a 30-bit prime $q = 2^{30} - 2^{18} - 1$. It shows the time one decoding takes in seconds with an increasing error t and also the average rate of a successful decoding with the same increasing error. The machine has a 3.2GHz single core with 8GB RAM. The figure shows that we can correct $O(\log n) = 10$ errors. The maximum number of tries for decoding is n.

decoding time increases with an increasing error t for $n = 256$, $m = 3n$ and $q = 1073479679$. We set a limit for the decoding algorithm at a maximum of n decoding attempts. As a result, no simulation runs longer than ca. 100 seconds. However, this speedup comes at the cost that not all tests can be successfully decoded, starting with an error of $t = 11$. Thus, with our implementation we can correct up to

$$\frac{t}{m} = \frac{10}{768} = 1.3\% \text{ word errors.} \tag{10}$$

We motivate use cases for this rather low error rate in Section 9. Note that decoding can be allowed more time to correct more errors and that the time our implementation takes is independent from the bit width of $\mathbf{w}, \mathbf{w}' \in \mathbb{F}_{\rho q}^m$.

8 Security Analysis

In this section we provide the necessary proofs and analysis to show the security of our system.

8.1 Security for Reverse and Robust CFE

For the sake of completeness we formally state that our construction is a computational fuzzy extractor.

Theorem 3 (Computational Fuzzy Extractor) *Our reverse computational fuzzy extractor as shown in Fig. 2 is an $(\mathcal{M}, W, m \log |\mathcal{Z}|, t)$-computational fuzzy extractor that is (ϵ, s_{sec})-hard with error $\delta = e^{-\Omega(n)}$ as in Definition 2.*

Proof. Our construction is based on the computational fuzzy extractor of Fuller *et al.* [20]. They are identical in terms of the high-level structure as well as the chosen parameters (see Section 2.3). Therefore our construction inherits the hardness properties stated in Theorem 2 and equally meets the requirements of Definition 2 for being a $(\mathcal{M}, W, m \log |\mathcal{Z}|, t)$-computational fuzzy extractor.

Our construction shown in Fig. 2 is also a reverse fuzzy extractor.

Theorem 4 (Reverse Fuzzy Extractor) *Our reverse computational fuzzy extractor as shown in Fig. 2 is an $(\mathcal{M}, m, n, t)$-reverse fuzzy extractor as in Definition 3.*

Proof (Sketch). The correctness property in Definition 3 corresponds to the correctness property of computational fuzzy extractors (see Definition 2). By Theorem 3 our construction is an $(\mathcal{M}, W, m \log |\mathcal{Z}|, t)$-computational fuzzy extractor that is (ϵ, s_{sec})-hard with error $\delta = e^{-\Omega(n)}$. Therefore, it is also correct in the sense of Definition 3.
The security property in Definition 3, on the other hand, ensures that the public helper data p generated by Gen does not reveal anything about the input $\mathbf{w}$ to the adversary.
Intuitively, p can be regarded as a public key and $\mathbf{w}$ as the corresponding secret key in an encryption scheme. In fact, Ben-Sasson *et al.* construct a generalized encryption scheme based on the LWE-assumption where the noise in the LWE-term is used as the secret key $e \in \mathbb{F}_q^m$ and the public key is constructed by choosing a random matrix $\mathbf{G}$ in $\mathbb{F}_q^{m \times n}$, selecting a random $\mathbf{g} \in \text{Image}(\mathbf{G})$ and computing $(\mathbf{G}, \mathbf{b} = \mathbf{g} + \mathbf{e})$ [6].
Looking at our construction, it is straightforward to see that $\mathbf{w}$ is analogue to $\mathbf{e}$ in Ben-Sasson *et al.*'s construction. Similarly, p can be expressed as the public key in their scheme: let $\text{Im}(M)$ be a function that takes a matrix $\mathbf{M} \in \mathbb{F}_q^{m \times n}$, chooses a random vector $\mathbf{x} \in \mathbb{F}_q^n$ and outputs $\mathbf{Mx}$.

$$\mathbf{G} := \mathbf{A}, \quad \mathbf{e} := \mathbf{w}, \quad \text{Image} := \text{Im} \tag{11}$$

$$p = (\mathbf{A}, \mathbf{Ax} + \mathbf{w} = \text{Im}(\mathbf{A}) + \mathbf{w}) = (\mathbf{G}, \text{Image}(\mathbf{G}) + \mathbf{e}) \tag{12}$$

Hence, the security of the reverse fuzzy extractor built from our construction corresponds to the hardness of recovering the secret key from the public key in Ben-Sasson *et al.*'s unified framework. The latter in turn is hard under the LWE-assumption, therefore our construction is a secure reverse fuzzy extractor under the assumption that LWE is hard, i.e. Theorem 1.

In fact our construction fulfills the stronger notion of security against outsider chosen perturbation attacks.

Theorem 5 (Outsider Chosen Perturbation Security) *Our reverse computational fuzzy extractor as shown in Fig. 2 is an $(\mathcal{M}, m, m' = m, t)$-reverse fuzzy extractor secure against outsider chosen perturbation attacks as in Definition 4.*

Proof. By Theorem 4 we know that a PPT adversary $\mathcal{A}$'s chance of recovering $\mathbf{w}$ from an individual output of the oracle in the outsider chosen perturbation security game (see Definition 4) is negligible in n. Therefore, we analyze the security implications of $\mathcal{A}$ requesting a set of perturbed outputs through the security game. Her available information will look like a system of v equations where v is the number of adversarial queries and $\mathbf{w}_i = \mathbf{w} + \mathbf{e}_i$:

$$\left\{(\mathbf{A}_i, \mathbf{b}_i = \mathbf{A}_i\mathbf{x}_i + \mathbf{w}_i) \leftarrow \mathsf{Gen}(\mathbf{w}_i)\right\}_{i\in\{1,\dots,v\}}. \tag{13}$$

Extracting the $\mathcal{A}$'s chosen perturbations $\mathbf{e}_i$, i.e. basic transformations $\mathbf{d}_i = \mathbf{b}_i - \mathbf{e}_i$ and $\mathbf{y}_i = \mathbf{A}_i\mathbf{x}_i$ give

$$\left\{(\mathbf{A}_i, \mathbf{d}_i = \mathbf{y}_i + \mathbf{w}) \leftarrow \mathsf{Gen}(\mathbf{w}_i)\right\}_{i\in\{1,\dots,v\}}. \tag{14}$$

In the resulting system of equations $\mathbf{y}_1, \dots, \mathbf{y}_v$ and $\mathbf{w}$ are unknowns, i.e. always one more unknown than given equations. Hence the adversary is presented with an underdetermined system of equations, which can either have no solutions or an infinite amount of solutions. Since $\mathbf{w}$ has been fixed beforehand we know that there must be at least one solution and therefore the number of all possible solutions is infinite over the domain $\mathcal{M}$.
Therefore, $\mathcal{A}$'s success probability in finding the correct solution $\mathbf{w}$ is negligible in m and hence our construction is an $(\mathcal{M}, m, m' = m, t)$-reverse fuzzy extractor secure against outsider chosen perturbation attacks.

As before, we formally state that our construction is a robust fuzzy extractor in Theorem 6 below. Note that we do not consider post-application robustness as defined in [16, Definition 6], where the adversary is provided with r from $(r, p) = \mathsf{Gen}(\mathbf{w}')$, as in our case r corresponds to the shared secret between device and server.

Theorem 6 (Robust Fuzzy Extractor) *Our reverse and robust computational fuzzy extractor as shown in Fig. 3 is an (ϵ, s_{sec})-hard $(|W|, k, t, \epsilon)$-fuzzy extractor with pre-application robustness $\mathsf{negl}(m)$ as in Definition 5.*

Proof (Sketch). As described above we add a Mac-tag σ to the output generated from Gen. σ is calculated over $p = (\mathbf{A}, \mathbf{A}\mathbf{x} + \mathbf{w}')$ using the secret $\mathbf{x}_{1,\dots,n}$ as key. In fact, recovering $\mathbf{x}_{1,\dots,n}$ from p corresponds to recovering the secret from an LWE-term, i.e. the search version of LWE [42]. Hence, given p, no efficient adversary can compute $\mathbf{x}_{1,\dots,n}$ except with negligible probability.
Therefore tampering with p i.e. producing p' will be detected, since the adversary cannot compute a valid σ' without knowledge of $\mathbf{x}_{1,\dots,n}$.
Concretely, $\mathsf{Ver}(\tilde{\sigma}, \tilde{\mathbf{A}}, (\widetilde{\mathbf{A}\mathbf{x} + \mathbf{w}'}), \mathbf{x}_{1,\dots,n})$ computes $\tilde{\mathbf{A}} \cdot \mathbf{x}_{1,\dots,n} - (\widetilde{\mathbf{A}\mathbf{x} + \mathbf{w}'}) = \tilde{\mathbf{w}}'$. Since by Theorem 4 it is hard to recover $\mathbf{w}'$ from p, $\mathsf{dis}(\tilde{\mathbf{w}}', \mathbf{w}) > t$ except with negligible probability in m. Therefore $\mathsf{Ver}(\tilde{p})$ will always reject if $\tilde{p} \neq p$ and cause $\mathsf{Rep}(\tilde{p})$ to output $\perp$.

8.2 Security of the Authentication Protocol

Correctness In the context of authentication *correctness* means that an honest party is always able to authenticate itself to an other honest party. If this relation holds both ways, we call it correct *mutual* authentication.

Definition 6 (Correct Mutual Authentication [49, Definition 4]) *A mutual authentication scheme between a device $\mathcal{D}$ and a server $\mathcal{S}$ is correct if an honest $\mathcal{D}$ always makes an honest $\mathcal{S}$ accept and vice versa.*

Theorem 7 (Correctness: Mutual Authentication) *Our mutual authentication scheme as shown in Fig. 4 is correct as in Definition 6.*

Proof. The correctness property of the reverse fuzzy extractor ensures that $\mathsf{Rep}(\mathbf{w}, p) = r$ where $(r, p) = \mathsf{Gen}(\mathbf{w}')$ as long as $\mathsf{dis}(\mathbf{w}, \mathbf{w}') \leq t$. The responses $\mathbf{w}'$ and $\mathbf{w}$ for an honest $\mathcal{D}$ and an honest $\mathcal{S}$ fulfill the distance requirement. Therefore $\mathcal{S}$ will always reconstruct the same $r = \mathbf{x}_{1,\dots,k}$ using Rep when provided with the helper data p from $\mathcal{D}$'s Gen. This implies both the acceptance of $\mathcal{D}$ by an honest $\mathcal{S}$ and the acceptance of S by an honest $\mathcal{D}$, since both depend on the correct calculation of p (on $\mathcal{D}$'s side), reconstruction of the secret r (on $\mathcal{S}$'s side) and leaving the secret unchanged throughout one protocol run (on both sides).

Device Authentication A device authentication mechanism is considered secure when no PPT adversary $\mathcal{A}$ succeeds in making an honest $\mathcal{S}$ accept $\mathcal{A}$ as a legitimate $\mathcal{D}$. The resulting *device authentication game* allows $\mathcal{A}$ to freely interact with an honest $\mathcal{S}$ and $\mathcal{D}$, record exchanged messages between them or manipulate messages in order to make $\mathcal{S}$ accept after a polynomial (in the chosen security parameter) number of adversarial queries.

Definition 7 (Device Authentication [49, Definition 5]) *An authentication scheme achieves μ-device authentication if any PPT-adversary wins the device authentication game with probability at most $\mathsf{negl}(\mu)$.*

Note that we only give a high-level description of the security notion, since the proof of security is in fact directly inherited from [49, Theorem 4].

Theorem 8 (Security: Device Authentication) *Our mutual authentication scheme as shown in Fig. 4 achieves $\mu = m$-device authentication in the random oracle model as in Definition 7 when using the reverse and robust fuzzy extractor as shown in Fig. 3.*

Proof (Sketch). According to [49] the existence of a PPT adversary $\mathcal{A}$ that wins the device authentication game with non-negligible advantage implies the existence of a PPT adversary $\mathcal{B}$ that has non-negligible advantage in winning the outsider chosen perturbation security game as in Definition 4. Since the structure of our authentication scheme as shown in Fig. 4 corresponds entirely to the authentication scheme in [49], we can reuse this result. I.e. by Theorem 5 there is no PPT adversary $\mathcal{B}$ that wins the outsider chosen perturbation security game with probability higher than $\mathsf{negl}(m)$. Therefore, from the non-existence of $\mathcal{B}$ follows by contraposition the non-existence of $\mathcal{A}$ with probability at least $1 - \mathsf{negl}(m)$.

Server Authentication A device authentication mechanism is considered secure when no PPT adversary $\mathcal{A}$ succeeds in making an honest $\mathcal{D}$ accept $\mathcal{A}$ as a legitimate $\mathcal{S}$. Analogous to the device authentication game (see 8.2), the *server authentication game* lets $\mathcal{A}$ conduct passive and active attempts in order to make an honest $\mathcal{D}$ accept $\mathcal{A}$ as a legitimate $\mathcal{S}$.

Definition 8 (Server Authentication [49, Definition 6]) *An authentication scheme achieves μ-server authentication if any PPT-adversary wins the server authentication game with probability at most* $\mathsf{negl}(\mu)$.

As before, we only give a high-level description of the security notion, since the proof of security is equivalent to the proof of secure server authentication in [49, Theorem 5].

Theorem 9 (Security: Server Authentication) *Our mutual authentication scheme as shown in Fig. 4 achieves $\mu = k$-server authentication in the random oracle model as in Definition 8 when using the reverse and robust fuzzy extractor as shown in Fig. 3.*

Proof (Sketch). Similar to the proof of Theorem 8 we use the fact that the existence of a PPT adversary $\mathcal{A}$ that wins the server authentication game with non-negligible advantage implies the existence of a PPT adversary $\mathcal{B}$ that has non-negligible advantage in winning the outsider chosen perturbation security game as in Definition 4 [49]. Since the structure of our authentication scheme as shown in Fig. 4 corresponds entirely to the authentication scheme in [49], we can reuse this result. I.e. by Theorem 5 there is no PPT adversary $\mathcal{B}$ that wins the outsider chosen perturbation security game with probability higher than $\mathsf{negl}(m)$. Therefore, from the non-existence of $\mathcal{B}$ follows by contraposition the non-existence of $\mathcal{A}$ with probability at least $1 - \mathsf{negl}(m)$. Note that a replay attack would only be successful if $\mathcal{D}$ selects the same secret $\mathbf{x}_{1,\ldots,k}$ twice, s.t. $\mathcal{B}$ can reuse a previously recorded b for the last message of the protocol. Since $\mathcal{D}$ selects $\mathbf{x}$ randomly in every round, the probability of a successful replay attack is negligible in the length of the secret k. Since $m > k$, the overall probability of adversary $\mathcal{A}$ succeeding in the server authentication game is therefore $\mathsf{negl}(k)$.

8.3 Computational Security and Related Work

Our CFE deals with two noises. The error t is from the pre-shared entropy source, measured in $\mathbf{w}$. The second noise, α, describes the error in the LWE problem making it hard, as τ for LPN and ρ for the lossless CFE, respectively. A lower α-noise allows a better decoding and a higher α-noise increases security. This is

Table 1: Table for different parameter setting for three security levels from the literature.

Security	**80-bit**	**128-bit**	**256-bit**
Herder *et al.* [26] (LPN)	– – – –	$n = 540$ $\tau = 0.4$ $(\alpha \approx 0.13698)$ $q = 2$	– – – –
Dagdelen *et al.* [11] (LWE)	$n = 240$ $\alpha \approx 0.00026$ $q = 327680$	$n = 320$ $\alpha \approx 0.00024$ $q = 327680$	$n = 550$ $\alpha \approx 0.00022$ $q = 327680$
Regev [42] (LWE)	$n = 160$ $\alpha \approx 0.00147$ $q = 25601$	$n = 256$ $\alpha \approx 0.00098$ $q = 65537$	$n = 480$ $\alpha \approx 0.00058$ $q = 230431$
LindnerPeikert [31] (LWE)	$n = 192$ $\alpha \approx 0.00217$ $q = 4093$	$n = 256$ $\alpha \approx 0.00205$ $q = 4093$	$n = 660$ $\alpha \approx 0.00168$ $q = 4093$
this work, lossless construction	$n = 256$ $\rho \geq 0.003072$ $(\alpha \geq 0.000004)$ $q = 1073479679$	$n = 256$ $\rho \geq 0.0384$ $(\alpha \geq 0.00005)$ $q = 1073479679$	$n = 256$ $\rho \geq 0.09984$ $(\alpha \geq 0.0009)$ $q = 1073479679$

why we state α and ρ as a lower bound in Table 1, it is the minimal noise we need from our pre-shared entropy source while still having, e.g., 128-bit security.

Herder *et al.* [26] give parameters for a security of 128 bit for their computational fuzzy extractor based on the LPN problem. LPN was introduced by Hopper and Blum [28] and has (n, τ) as parameters. Herder *et al.* conclude that their construction meets the required security with parameters $n = 540$ and $\tau = 0.4$ with respect to the attack by Bernstein and Lange [8]. We find, while maintaining 128 bit of security, these parameters translate to $n = 540$, $\alpha = 0.13698$ and $q = 2$ in the LWE problem with the estimator from Albrecht *et al.* [2]. Note that the modulus q is inherently 2 in LPN. Also, Herder *et al.* implemented their fuzzy extractor on a resource rich computer, so there was no need for optimization for an embedded environment.

Dagdelen *et al.* [11] give parameter settings for three different security levels. We also compare our work with Regev's example choices for parameter [42], i.e., $q \approx n^2$ and $\alpha = 1/(\sqrt{2\pi n}\log_2^2 n)$. Lindner and Peikert [31] chose a relatively small modulus q, while increasing the noise level α for their optimized construction.

In 2015, Albrecht *et al.* [2] published their work on the hardness of the LWE problem, collecting and estimating parameters for different attack algorithms. They surveyed lattice reduction, strategies, attacks and estimation algorithms, e.g., [36,37,38,31,9]. We used their estimator to compare the needed number of bit operations for a successful attack. However, our problem is based on a m-bounded $\mathsf{dist\text{-}LWE}_{(n,m,q,\chi)}$ problem, whereas the estimations assume an unlimited amount of elements w_i, meaning m is unbounded. Nevertheless we used the estimator, because Lindner and Peikert [31] state that for sufficiently large m, with $m \geq 200$, the number of rows m in the matrix $\mathbf{A}$ becomes irrelevant.

We collect parameter values from the literature in Table 1. The parameters are the dimension n, the noise size α and the modulus q. Note that all parameters n, α and q increase the security. We estimate the security of our implementation with a given n and q and a needed minimum noise level α. The Security of our $\mathsf{dist\text{-}LWE}_{(n,m,q,\mathcal{U}([-\rho q,\rho q]))}$ problem has ρ as parameter, but we know it is at least as hard as $\mathsf{dist\text{-}LWE}_{(n,m,q,\Psi_\alpha)}$ [18]. Also due to Döttling and Müller-Quade, we know that $\alpha = \frac{\rho}{mn^\sigma}$, with $\sigma \in (0,1)$. We further assume, as in [11] that computers execute about 2^{10} operations per second.

Information-theoretically secure FE deal with PUF bit error rates ranging from ca. 1% to 15% [35,29]. BCH codes are popular for correcting these PUFs [15] and robustness can be increased by, e.g., an interleaved code [22,3]. For comparison, a (n,k,t)-BCH code has a decoding complexity of $O(nt)$ [45]. Clearly, traditional FEs can tolerate more errors, compared to our 1.3% *word* errors, and have a smaller decoding complexity, but they always have an entropy loss. When focusing on low-noise entropy sources and relaxed time constrains, our CFE is a no entropy loss alternative, even for lightweight IoT systems.

9 Potential Pre-shared Entropy Sources

A specific PUF construction is not relevant as long as the error parameters of the CFE can be combined with a good use case, e.g., biometrics, passwords or PUFs. Nevertheless, we provide several practical use cases for our lossless construction, matching the error constraint given in (10).

9.1 Passwords

Our construction excels at sources of entropy where the error occurs as a representation of multiple bits, rather than a single bit flip. This means that whether just a single bit is flipped in the vector element w_i or all bits in w_i, it counts as one error.

A suitable problem for this type of error are passwords. For Construction 1 a password is vector $\mathbf{w}$ and r is the retrieved key. Here, we can allow an user to enter up to t wrong symbols and key derivation or authentication would still be correct. An incorrect symbol, e.g., represented by a char, results in the desired error type, refraining from a pure bit representation. One symbol would be represented by one element in $\mathbf{w}$ and the symbols have to be chosen at uniform to hold Theorem 2. However, in this use case we are able to correct $O\log(n)$ errors efficiently and security still relies computationally on $\mathbf{w}$. Again, we emphasize that the lost entropy is zero, i.e., $H(w) - H(r) = 0$.

9.2 Low noise PUFs

The parameters of the lossless construction do not provide error correcting capabilities to work with the error rate of every PUF. For our system to work, we need low noise PUFs. An example of such PUFs is the V_T-PUF proposed by Lofstrom *et al.* [32] with an error rate of 1.3%. Another example is a ring oscillator PUF (RO-PUF) with reliability enhancement by Suh and Devadas [47] with an error rate of 0.48%.

9.3 RO-PUF with Trapdoor

One way to improve the decoding is the introduction of a trapdoor. Herder *et al.* [26] proposed a Trapdoor CFE working with a RO-PUF, where the trapdoor is the PUF itself. Their PUF outputs an additional confidence information alongside with the response and based on this confidence information the more robust values of $\mathbf{w}$ are chosen first for decoding. Our algorithms become more efficient by adapting the constructions of [26] in a straightforward manner. If doing this, the results will not change in terms of size requirements, only decoding performance changes for the better due to provided confidence information.

9.4 Randomness Extraction before Error Correction

Another idea is to extract the randomness of $\mathbf{w}$ *before* the error correction. For this, we split our source $\mathbf{w}$ into m blocks with an arbitrary length. Then each block w_i gets reduced within a privacy amplification step Ext to a bit width ρq and each element w_i is then close to uniform. The idea is sketched in (15) in Appendix C. Note that privacy amplification or randomness extraction does not hold the distance between these blocks, meaning $\mathsf{dis}(w_i, w_i') \neq \mathsf{dis}(w_i, \mathsf{Ext}(w_i'))$, but this is just the case where the computational fuzzy extractor happens to be good at. Error correction, i.e., decoding of a random linear code, on high entropy words can be done more efficient. Also, this construction enables the input from sources with different length.

10 Conclusion

We showed how a client-server authentication system can be secured with a computational fuzzy extractor while providing outsider chosen perturbation security and pre-application robustness.

We presented the first actual implementation of a lossless computational fuzzy extractor in an authentication system, thereby bridging the gap between the CFE theory and practice. Our implementation needs 1.45KB of SRAM and 9.8KB of Flash memory and runs on the low-cost Atmel 8-bit AVR RISC-based microcontroller in 34.9 seconds. The CFE needs 0.4 seconds, when implemented on a 32-bit IoT device. Our implementation has a (very) small memory footprint due to the optimizations we performed on the original CFE algorithms presented by Fuller *et al.* in [20]. As a side effect, the resource efficiency of our implementation demonstrates the feasibility of computational fuzzy extractors for resource-constrained ecosystems like the Internet of Things.

We discussed the relations between the parameters of lossy and lossless CFE constructions as well as the conditions the parameters must satisfy to ensure the security of CFE schemes. Based on these considerations, we selected an exemplary set of CFE parameter values that satisfies the constraints of our hardware platform.

Finally, we compared our results to existing work under different security assumptions related to the available noise level.

References

1. Akavia, A., Goldwasser, S., Vaikuntanathan, V.: Simultaneous hardcore bits and cryptography against memory attacks. In: Theory of Cryptography, pp. 474–495. Springer (2009)
2. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology 9(3), 169–203 (2015)
3. Aysu, A., Gulcan, E., Moriyama, D., Schaumont, P., Yung, M.: End-to-end design of a puf-based privacy preserving authentication protocol. In: Cryptographic Hardware and Embedded Systems–CHES 2015, pp. 556–576. Springer (2015)
4. Bandyopadhyay, D., Sen, J.: Internet of things: Applications and challenges in technology and standardization. Wireless Personal Communications 58(1), 49–69 (2011)
5. Barker, E., Kelsey, J.: Nist draft special publication 800-90b recommendation for the entropy sources used for random bit generation (2012)
6. Ben-Sasson, E., Ben-Tov, I., Damgard, I., Ishai, Y., Ron-Zewi, N.: On public key encryption from noisy codewords. In: PKC 2015. vol. 9615, pp. 417–446 (2015)
7. Bennett, C.H., Brassard, G.: Quantum cryptography: Public key distribution and coin tossing. In: IEEE International Conf. on Computers, Systems and Signal Processing (Bangalore). pp. 175–179 (1984)
8. Bernstein, D.J., Lange, T.: Never trust a bunny. In: Radio Frequency Identification. Security and Privacy Issues, pp. 137–148. Springer (2012)
9. Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. Journal of the ACM (JACM) 50(4), 506–519 (2003)
10. Boyen, X., Dodis, Y., Katz, J., Ostrovsky, R., Smith, A.: Secure remote authentication using biometric data. In: Advances in cryptology–EUROCRYPT 2005, pp. 147–163. Springer (2005)
11. Dagdelen, Ö., Gajek, S., Göpfert, F.: Learning with errors in the exponent. IACR Cryptology ePrint Archive 2014, 826 (2014)
12. Daugman, J.: How iris recognition works. Circuits and Systems for Video Technology, IEEE Transactions on 14(1), 21–30 (2004)
13. Delvaux, J., Peeters, R., Gu, D., Verbauwhede, I.: A survey on lightweight entity authentication with strong pufs. ACM Computing Surveys (CSUR) 48(2), 26 (2015)
14. Delvaux, J., Verbauwhede, I.: Attacking puf-based pattern matching key generators via helper data manipulation. In: Topics in Cryptology–CT-RSA 2014, pp. 106–131. Springer (2014)

15. Devadas, S., Yu, M.D.: Secure and robust error correction for physical unclonable functions (2013)
16. Dodis, Y., Kanukurthi, B., Katz, J., Reyzin, L., Smith, A.: Robust fuzzy extractors and authenticated key agreement from close secrets. Information Theory, IEEE Transactions on 58(9), 6207–6222 (2012)
17. Dodis, Y., Reyzin, L., Smith, A.: Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In: Advances in cryptology-Eurocrypt 2004. pp. 523–540. Springer (2004)
18. Döttling, N., Müller-Quade, J.: Lossy codes and a new variant of the learning-with-errors problem. In: Advances in Cryptology–EUROCRYPT 2013, pp. 18–34. Springer (2013)
19. Evans, D.: The internet of things: How the next evolution of the internet is changing everything. CISCO white paper 1, 1–11 (2011)
20. Fuller, B., Meng, X., Reyzin, L.: Computational fuzzy extractors. In: Sako, K., Sarkar, P. (eds.) Advances in Cryptology — ASIACRYPT 2013(1). LNCS, vol. 8269, pp. 174–193. Springer (December 1-5, 2013)
21. Gassend, B., Clarke, D.E., van Dijk, M., Devadas, S.: Silicon physical random functions. In: Atluri, V. (ed.) Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002. pp. 148–160. ACM (2002)
22. Gassend, B.L.: Physical random functions. Ph.D. thesis, Massachusetts Institute of Technology (2003)
23. Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.: Physical unclonable functions and public-key crypto for fpga ip protection. In: Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on. pp. 189–195. IEEE (2007)
24. Guajardo, J., Kumar, S., Schrijen, G.J., Tuyls, P.: Fpga intrinsic pufs and their use for ip protection. In: Paillier, P., Verbauwhede, I. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2007, Lecture Notes in Computer Science, vol. 4727, pp. 63–80. Springer Berlin Heidelberg (2007)
25. Gura, N., Patel, A., Wander, A., Eberle, H., Shantz, S.C.: Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings, chap. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs, pp. 119–132. Springer Berlin Heidelberg, Berlin, Heidelberg (2004), http://dx.doi.org/10.1007/978-3-540-28632-5_9
26. Herder, C., Ren, L., van Dijk, M., Yu, M.D.M., Devadas, S.: Trapdoor computational fuzzy extractors. IACR Cryptology ePrint Archive 2014, 938 (2014)
27. Holcomb, D.E., Burleson, W.P., Fu, K.: Power-up sram state as an identifying fingerprint and source of true random numbers. Computers, IEEE Transactions on 58(9), 1198–1210 (2009)
28. Hopper, N.J., Blum, M.: Secure human identification protocols. In: Advances in cryptology - ASIACRYPT 2001, pp. 52–66. Springer (2001)
29. Katzenbeisser, S., Kocabaş, Ü., Rožić, V., Sadeghi, A.R., Verbauwhede, I., Wachsmann, C.: Pufs: Myth, fact or busted? a security evaluation of physically unclonable functions (pufs) cast in silicon. In: Cryptographic Hardware and Embedded Systems–CHES 2012, pp. 283–301. Springer (2012)
30. Kristinsson, B.: Ardrand: The arduino as a hardware random-number generator (2011)
31. Lindner, R., Peikert, C.: Better key sizes (and attacks) for lwe-based encryption. In: Topics in Cryptology–CT-RSA 2011, pp. 319–339. Springer (2011)

32. Lofstrom, K., Daasch, W.R., Taylor, D.: Ic identification circuit using device mismatch. In: Solid-State Circuits Conference, 2000. Digest of Technical Papers. ISSCC. 2000 IEEE International. pp. 372–373. IEEE (2000)
33. Maes, R.: Physically unclonable functions: Constructions, properties and applications (fysisch onkloonbare functies: constructies, eigenschappen en toepassingen) (2012)
34. Maes, R., van der Leest, V., van der Sluis, E., Willems, F.: Secure key generation from biased pufs. In: Cryptographic Hardware and Embedded Systems–CHES 2015, pp. 517–534. Springer (2015)
35. Maes, R., Verbauwhede, I.: Physically unclonable functions: A study on the state of the art and future research directions. In: Towards Hardware-Intrinsic Security, pp. 3–37. Springer (2010)
36. Micciancio, D., Regev, O.: Lattice-based cryptography. In: Post-quantum cryptography, pp. 147–191. Springer (2009)
37. Nguyen, P.Q.: Hermite's constant and lattice algorithms. In: The LLL Algorithm, pp. 19–69. Springer (2009)
38. Nguyen, P.Q.: Lattice reduction algorithms: Theory and practice. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 2–6. Springer (2011)
39. NIST, S.: Special publication 800-22. A statistical test suite for random and pseudorandom number generators for cryptographic applications (2010)
40. Pappu, S.R.: Physical one-way functions. Ph.D. thesis, Massachusetts Institute of Technology (2001)
41. Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem. In: Proceedings of the forty-first annual ACM symposium on Theory of computing. pp. 333–342. ACM (2009)
42. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) 56(6), 34 (2009)
43. Regev, O.: The learning with errors problem. Invited survey in CCC (2010)
44. Schaller, A., Katzenbeisser, S.: Eliminating leakage in reverse fuzzy extractors.
45. Schipani, D., Elia, M., Rosenthal, J.: On the decoding complexity of cyclic codes up to the bch bound. In: Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on. pp. 835–839. IEEE (2011)
46. Solinas, J.A.: Generalized mersenne numbers. Citeseer
47. Suh, G.E., Devadas, S.: Physical unclonable functions for device authentication and secret key generation. In: Proceedings of the 44th annual Design Automation Conference. pp. 9–14. ACM (2007)
48. Suh, G., Devadas, S.: Physical unclonable functions for device authentication and secret key generation. In: Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE. pp. 9–14 (June 2007)
49. Van Herrewege, A., Katzenbeisser, S., Maes, R., Peeters, R., Sadeghi, A.R., Verbauwhede, I., Wachsmann, C.: Reverse fuzzy extractors: Enabling lightweight mutual authentication for puf-enabled rfids. In: Financial Cryptography and Data Security, pp. 374–389. Springer (2012)

A Detailed Memory Footprint

Table 2 gives a detailed memory footprint of our reverse and robust lossless computational fuzzy extractor implementation on the device (Gen procedure). Our own files are marked with an *.

Table 2: Memory footprint of on the device.

file	**.text (byte)**	**.data (byte)**	**.bss (byte)**	**total (byte)**
`SampleX`*	38	0	0	38
`modularreduction`*	582	0	0	582
`getrand`*	128	0	0	128
`add`*	230	0	0	230
`multiplication`*	228	0	0	228
`sha256`*	3060	0	297	3357
`Generate`*	438	0	1071	1509
total own files	4704	0	1368	6072
`abi`	8	0	0	8
`CDC`	576	8	80	664
`HardwareSerial`	752	0	0	752
`HardwareSerial1`	332	0	157	489
`HID`	1094	2	13	1109
`hooks`	2	0	0	2
`IPAddress`	320	0	6	326
`main`	46	0	0	46
`new`	16	0	0	16
`Print`	1620	0	0	1620
`Stream`	1381	0	0	1381
`Tone`	1509	1	42	1552
`USBCore`	1998	0	9	2007
`WInterrupts`	670	0	10	680
`wiring`	514	0	9	523
`wiring_analog`	436	1	0	437
`wiring_digital`	600	0	0	600
`wiring_pulse`	266	0	0	266
`wiring_shift`	232	0	0	232
`WMath`	298	0	0	298
`WString`	4747	0	1	4748
total auxiliary files	17417	327	297	17756

Table 3 gives the overall memory footprint of our reverse and robust lossless computational fuzzy extractor implementation on the device (Gen procedure).

Table 3: Overall memory footprint on the device.

AVR memory usage on device atmega32u4	
Program:	9840 bytes (30.0% .text+.data+.bootloader)
Data:	1451 bytes (56.7% .data+.bss+.noinit)

B Detailed Timing Results on Device

Table 4 provides detailed Gen timing results on an Atmega32u4 device. Note that the generation of all random numbers via TRG would be infeasible, so we use it as seed for a PRG.

Table 4: Detailed Timing Results on Device.

function name	**single call time (ms)**	**total time (ms)**
`multiplication`	0.045	8847
`modularreduction`	0.015	2949
`add`	0.003	590
`sha256`	34.168	34
`getrand` (PRG)	0.094	18481
`getrand` (TRG)	(41.836)	(8225292)
remaining overhead	–	3983
`Generate`	–	34885

C Sketched Randomness Extraction before Error Correction

In (15) the idea is sketched of extracting the randomness of $\mathbf{w}$ *before* the error correction. The source $\mathbf{w}$ is split into m blocks with an arbitrary length (here, $i, j, l, \ldots$). Then privacy amplification Ext reduces each block w_i to a bit width ρq, so that each element w_i is close to uniform.

$$\mathbf{w} \to \begin{pmatrix} \overbrace{w_1}^{\log i} \\ \vdots \\ \overbrace{w_k}^{\log j} \\ \vdots \\ \overbrace{w_m}^{\log l} \end{pmatrix} \to \begin{pmatrix} \overbrace{\mathsf{Ext}(w_1)}^{\log \rho q} \\ \vdots \\ \overbrace{\mathsf{Ext}(w_k)}^{\log \rho q} \\ \vdots \\ \overbrace{\mathsf{Ext}(w_m)}^{\log \rho q} \end{pmatrix} \tag{15}$$