Robust Fuzzy Extractors and Helper Data Manipulation Attacks Revisited: Theory vs Practice

Georg T. Becker

Abstract—Fuzzy extractors have been proposed in 2004 by Dodis *et al.* as a secure way to generate cryptographic keys from noisy sources. In recent years, fuzzy extractors have become an important building block in hardware security due to their use in secure key generation based on Physical Unclonable Functions (PUFs). Fuzzy extractors are provably secure against passive attackers. A year later Boyen *et al.* introduced robust fuzzy extractors which are also provably secure against active attackers, i.e., attackers that can manipulate the helper data.

In this paper we show that the original provable secure robust fuzzy extractor construction by Boyen *et al.* actually does not fulfill the error-correction requirements for practical PUF applications. The fuzzy extractors proposed for PUF-based key generation on the other hand that fulfill the error-correction requirements cannot be extended to such robust fuzzy extractors, due to a strict bound t on the number of correctable errors. While it is therefore tempting to simply ignore this strict bound, we present novel helper data manipulation attacks on fuzzy extractors that also work if a "robust fuzzy extractor-like" construction without this strict bound is used.

Hence, this paper can be seen as a call for action to revisit this seemingly solved problem of building robust fuzzy extractors. The new focus should be on building more efficient solutions in terms of error-correction capability, even if this might come at the costs of a proof in a weaker security model.

Index Terms—Robust Fuzzy Extractor, Physical Unclonable Functions (PUFs), Helper Data Manipulation Attacks

1 Introduction

Fuzzy extractors have been proposed in 2004 by Dodis *et al.* [10] as a provably secure way to generate cryptographic keys from correlated but possibly noisy sources. The main motivation for this work back in 2004 were biometrics: During a generation phase, a biometric reading such as a fingerprint or iris picture is used to generate a cryptographic key and helper data. At a later time, a second, possibly noisy, reading of the same biometric source is used in conjunction with the helper data to reproduce the cryptographic key. Fuzzy extractors guarantee that *i)* the derived key is uniform even after revealing the helper data, i.e., the helper data does not reveal information about the key, and *ii)* as long as the distance between the two readings is smaller than or equal to a constant t, the same key is extracted during the recreation step.

The author is with the Digital Society Institute at the ESMT Berlin, Germany. Most of the work was done while he was with the Horst Görtz Institute for IT Security, Ruhr-Universität Bochum, Germany. E-mail: {firstname.lastname}@rub.de.

The author would like to thank Johannes Tobisch for providing the GMC decoding implementation and for his helpful comments as well as the anonymous reviewers for their helpful feedback.

While fuzzy extractors are provably secure against passive attackers, this does not say anything about attackers who can alter the helper data. Therefore, Boyen *et al.* proposed a construct called *robust fuzzy extractor* in 2005 [2] which is also secure against helper data manipulation attacks. The original robust fuzzy extractor was only provably secure in the random oracle model, but a provably secure construction in the general model was proposed a year later in [8] and slightly improved in [7], [14]. It is noteworthy that, in theory, any fuzzy extractor can be turned into a robust fuzzy extractor. Hence, at least for the Hamming distance metric, the problem of building provably secure fuzzy extractors and provably secure robust fuzzy extractors seems to be solved.

From a practical perspective, fuzzy extractors have become increasingly important not due to biometrics, but due to key generation based on Physical Unclonable Functions (PUFs). Storing and generating cryptographic keys in embedded devices can be a challenging task, in particular if they need to be able to withstand physical attacks. Standard non-volatile memory has little protection against an unauthorized read-out, especially if the non-volatile memory is not on the same chip. Only dedicated secure non-volatile memory can offer protection against an advanced attacker. PUFs are a promising alternative to such secure non-volatile memory. The idea of PUFs is to use the process variations within each chip to derive a unique "fingerprint", which is called the PUF response. A Fuzzy Extractor is then used to derive a cryptographic key and helper data from this PUF response. In the recovery phase, this helper data is used in conjunction with a (possibly noisy) PUF response to recover the key again. PUF-based key generation, and in particular how to construct fuzzy extractors for PUF-based key storage, has been the focus of a lot of research. One important focus of this research is how to handle the significant level of noise that can be present in the PUF responses. Several error-correcting codes and decoding strategies have been proposed to be used in fuzzy extractors for PUF-based key generation, e.g. in [1], [16], [17], [18], [19], [25]. It is noteworthy that fuzzy extractors and PUF-based key generation are not purely academic research topics anymore. Several high-security products by companies such as Microsemi, Altera and NXP use fuzzy extractors in conjunction with PUF-based key generation[1].

The first helper data manipulation attacks against PUF based key generation were not on fuzzy extractors, but on helper data algorithm based on pattern matching [5] and designs specifically for Ring-Oscillators [6]. Another helper data manipulation attack was presented on soft-decision error-correction [3]. While the attack was described based on the helper data algorithm by Maes *et al.* [18], which is strictly speaking not a fuzzy extractor as no bound on the min-entropy is provided, the attack can also be applied to even number repetition codes and hence fuzzy extractors.

In practice, most solutions for PUF-based generation need a robust fuzzy extractor, which has been acknowledged in numerous papers. However, papers describing actual implementations only used fuzzy extractors and not robust fuzzy extractors.

1.1 Main contribution

In this paper we will show that how to build provably secure as well as practical robust fuzzy extractors is not a solved problem. In particular, we show that:

- It is actually not possible to build a provably secure robust fuzzy extractor based on the Bose-Chaudhuri-Hocquenghem-Codes (BCH) construction proposed by Boyen *et al.* [2] for noise levels above 3%.
- The concatenated code constructions used in practical fuzzy extractor implementations that can handle noise levels of 15% or more are actually not well-formed and therefore using them in Boyen *et al.* 's construction violates the security proof.
- We introduce a new helper data manipulation attack strategy on linear codes which we demonstrate based on several decoding strategies for Reed–Muller codes and soft-decision decoding.
- In this new attack strategy the attacker *sets* a key as opposed to recovering a key. Therefore, the new attack strategy can be used to attack a robust fuzzy extractor-like construction which uses a hash function to check the integrity of the helper data, but ignores the strict bound t of the proof by Boyen *et al.* .

Finally, we also propose some promising research directions to solve this problem in the future.

1.2 Outline

The next Sections provides some necessary background information. In particular, robust fuzzy extractors and their related constructions are formally defined and introduced. In Section 3 the robust fuzzy extractor based on BCH codes is examined in greater detail, and it is shown that building a robust fuzzy extractor based on BCH codes is not possible for realistic parameters. Section 4 introduces the attack model and strategy of our new helper data manipulation attack and Section 5 presents some concrete examples of how the attack works against Reed–Muller codes and soft-decision decoding. Finally, a discussion outlook is provided in Section 6 with promising future research directions, before the paper finishes with a short conclusion.

1. NXP announced that the upcoming SmartMX2 security chips will feature a PUF, Altera uses PUFs in their Stratix 10 FPGAs, Microsemi in their SmartFusion FPGAs. All these products use Intrinsic-ID's licenses and PUF solutions.

2 Background

This Section introduces the formal definitions of robust fuzzy extractors and their building blocks. The corresponding proofs and a more detailed description can be found in the referenced papers. In the following we will use the definitions from [2].

2.1 Notations

In this paper we will use the following notations: A vector is represented with a bold lowercase character, a matrix with a bold uppercase character, a scalar with a lowercase character, a random variable with an uppercase character, and a set with calligraphic character. Table 1 is a summary of the variable names used in this paper to describe the robust fuzzy extractor for PUF based key generation and the helper data manipulation attack. In this context, a variable used in the recovery phase before the error-correction is denoted with a tick mark, a variable after the error-correction is denoted with a bar, and a value chosen or predicted by the attacker is indicated with an a index. For binary error-correction codes we use to the notion of $[n, k, d]$, with n being the codeword length, k being the dimension and d denoting the minimum distance between codewords.

	Original	Before Error Corr.	After Error Corr.	Attacker's choice
PUF response	$\mathbf{w}$	$\mathbf{w}'$	$\bar{\mathbf{w}}$	$\bar{\mathbf{w}}_a$
codeword	$\mathbf{x}$	$\mathbf{x}'$	$\bar{\mathbf{x}}$	$\bar{\mathbf{x}}_a$
helper data	$\mathbf{s}$			$\mathbf{s}_a$
hash value	$\mathbf{h}$		$\bar{\mathbf{h}}$	$\bar{\mathbf{h}}_a$
secret key	$\mathbf{r}$		$\bar{\mathbf{r}}$	$\bar{\mathbf{r}}_a$
error vector	$\mathbf{e}$			
attack vector				$\mathbf{e}_a$

TABLE 1: Overview of variable definitions

2.2 Definitions of secure sketches and fuzzy extractors

Fuzzy extractors and secure sketches were proposed by Dodis *et al.* in 2004 [10]. A secure sketch is defined as follows: [2]

Definition 1: An (m, m', t)-secure sketch over a metric space $(\mathcal{M}, \mathsf{d})$ comprises a *sketching procedure* $\mathsf{SS} : \mathcal{M} \to \{0,1\}^*$ and a *recovery procedure* Rec, where:

1) **(Security)** For all random variables W over $\mathcal{M}$ such that $\mathsf{H}_\infty(W) \geq m$, we have $\mathsf{H}_\infty(W|\mathsf{SS}(W)) \geq m'$, with H_∞ denoting the min-entropy.
2) **(Error tolerance)** For all pairs of points $\mathbf{w}, \mathbf{w}' \in \mathcal{M}$ with $\mathsf{d}(\mathbf{w}, \mathbf{w}') \leq t$, it holds that $\mathsf{Rec}(\mathbf{w}', \mathsf{SS}(\mathbf{w})) = \mathbf{w}$.

In the context of PUFs, a secure sketch can be used to generate helper data $\mathbf{s}$ for a PUF response $\mathbf{w}$ during the first key generation (sketching procedure). This helper data can then be used to correct up to t errors in a noisy PUF response $\mathbf{w}'$ during the "recovery" phase. The security guarantees state that the remaining min-entropy in the PUF response $\mathbf{w}$ after revealing the helper data $\mathbf{s}$ is at least $m' \leq m$. Hence, $\mathbf{w}$ is not guaranteed to be uniformly distributed and revealing the helper data reduces the guaranteed min-entropy. Therefore, $\mathbf{w}$ cannot be used directly as a cryptographic key. The main purpose of a Fuzzy Extractor is to extend the secure sketch to generate a string $\mathbf{r}$ that is uniformly distributed so that it can be be used as a cryptographic key. The formal definition is as follows: [2]

Definition 2: An (m, l, t, δ)-fuzzy extractor over a metric space $(\mathcal{M}, \mathsf{d})$ comprises a (randomized) *extraction algorithm* $\mathsf{Ext} : \mathcal{M} \rightarrow \{0,1\}^l \times \{0,1\}^*$ and a *recovery procedure* Rec such that:

1) **(Security)** For all random variables W over $\mathbf{M}$ that satisfy $\mathsf{H}^\infty(W) \geq m$, if $\langle \mathbf{r}, \mathbf{pub} \rangle \leftarrow \mathsf{Ext}(W)$ then $\mathsf{SD}(\langle \mathbf{r}, \mathbf{pub} \rangle, \langle \mathcal{U}_l, \mathbf{pub} \rangle) \leq \delta$, where $SD()$ is the statistical difference as defined in [2] and $\mathcal{U}_l$ is a random variable that is uniformly distributed over $\{0,1\}^l$.
2) **(Error tolerance)** Let $\mathsf{d}()$ be a distance metric for $\mathcal{M}$. Then for all pairs of points $\mathbf{w}, \mathbf{w}' \in \mathcal{M}$ with $\mathsf{d}(\mathbf{w}, \mathbf{w}') \leq t$, if $\langle \mathbf{r}, \mathbf{pub} \rangle \leftarrow \mathsf{Ext}(\mathbf{w})$ then it is the case that $\mathsf{Rec}(\mathbf{w}', \mathbf{pub}) = \mathbf{r}$.

The aforementioned fuzzy extractor is only secure against a passive attacker. I.e., it does not make any guarantees about the security if the attacker can manipulate the helper data $\mathbf{pub}$[2]. For this purpose, robust fuzzy extractors were proposed by Boyen *et al.* [2] that are also secure against active attackers. These robust fuzzy extractors are based on well-formed secure sketches which are defined as follows: [2]

Definition 3: An (m, m', t)-secure sketch $(\mathsf{SS}, \mathsf{Rec})$ is said to be well-formed if it satisfies the condition of Definition 1, except for the following modifications:

1) Rec may now return either an element in $\mathcal{M}$ or the distinguished symbol $\perp$.
2) For all $\mathbf{w}' \in \mathcal{M}$ and arbitrary $\mathbf{pub}'$, if $\mathsf{Rec}(\mathbf{w}', \mathbf{pub}') \neq \perp$ then $\mathsf{d}(\mathbf{w}', \mathsf{Rec}(\mathbf{w}', \mathbf{pub}')) \leq t$.

In other words, a secure sketch only guarantees that if there are t or less errors, the errors will be corrected. It does not say anything about what happens if there are more than t errors. Correcting more than t errors does not have any impact on security. In addition, t does not even need to be a strict lower bound but can be relaxed, which was formalized in [9] as *relaxed notions of correctness*. A well-formed secure sketch on the other hand corrects exactly up to t errors and responds with $\perp$ if there are more than t errors. One cannot simply relax this notion and allow more than t errors to be corrected without violating the assumptions in the proof. It is actually simple to construct a well-formed secure sketch from any secure sketch: For this, one only needs to compute $t' = \mathsf{d}(\mathbf{w}, \mathbf{w}')$. If $t' > t$ return $\perp$ else return $\mathbf{w}$.

2. With $\mathbf{pub}$ we denote the helper data to be in line with the definition in with [2]. The helper data is typically a vector $\mathbf{s}$. However, the exact format of the helper data is not defined and can be of any data type.

A *robust sketch* is resistant to helper data manipulation attacks and is defined as follows: [2]

Definition 4: Given algorithms $(\mathsf{SS}, \mathsf{Rec})$ and random variables $W = \{W_0, W_1, ..., W_n\}$ over the metric space $(\mathcal{M}, \mathsf{d})$, consider the following game between an adversary A and a challenger: Let $\mathbf{w}$ (resp., $\mathbf{w}_i$) be the value assumed by W_0 (resp., W_i). The challenger computes $\mathbf{pub} \leftarrow \mathsf{SS}(\mathbf{w}_0)$ and gives $\mathbf{pub}$ to A. Next, for $i = 1, ..., n$, the adversary A outputs a "challenge" $\mathbf{pub}_i \neq \mathbf{pub}$ and is given $\mathsf{Rec}(\mathbf{w}_i, \mathbf{pub}_i)$ in return. If there exists an i such that $\mathsf{Rec}(\mathbf{w}_i, \mathbf{pub}_i) \neq \perp$ we say that the adversary succeeds and this event is denoted by $Succ$.

We say that $(\mathsf{SS}, \mathsf{Rec})$ is an (m, m'', n, ϵ, t)-robust sketch over $(\mathcal{M}, \mathsf{d})$ if it is a well-formed (m, m'', t)-secure sketch and:

1) For all t-bounded distortion ensembles W with $\mathsf{H}_\infty(W_0) \geq m$ and all adversaries A we have $Pr[Succ] \leq \epsilon$.
2) The average min-entropy of W_0, conditioned on the entire view of A throughout the above game, is at least m''. Which implies that $(\mathsf{SS}, \mathsf{Rec})$ is an (m, m'', t)-secure sketch.

Similar to secure sketches, a robust sketch does not necessarily produce a uniformly distributed string that can be used as a cryptographic key. For this purpose robust fuzzy extractors were defined as follows: [2]

Definition 5: Given algorithms $(\mathsf{Ext}, \mathsf{Rec})$ and random variables $W = \{W_0, W_1, ..., W_n\}$ over a metric space $(\mathcal{M}, \mathsf{d})$, consider the following game between an adversary A and a challenger: Let $\mathbf{w}_0$ (resp., $\mathbf{w}_i$) be the value assumed by W_0 (resp., W_i). The challenger computes $(\mathbf{r}, \mathbf{pub}) \leftarrow \mathsf{Ext}(\mathbf{w}_0)$ and gives $\mathbf{pub}$ to A. Next, for $i = 1, .., n$, the adversary A outputs $\mathbf{pub}_i \neq \mathbf{pub}$ and is given $\mathsf{Rec}(\mathbf{w}_i, \mathbf{pub}_i)$ in return. If there exists an i such that $\mathsf{Rec}(\mathbf{w}_i, \mathbf{pub}_i) \neq \perp$ we say the adversary succeeds and this event is denoted by $Succ$. We say $(\mathsf{Ext}, \mathsf{Rec})$ is an $(m, l, n, \epsilon, t, \delta)$-robust fuzzy extractor over $(\mathcal{M}, \mathsf{d})$ if the following hold for all t-bounded distortion ensembles W with $\mathsf{H}_\infty(W_0) \geq m$:

- **(Robustness)** For all adversaries A, it holds that $Pr[Succ] \leq \epsilon$.
- **(Security)** Let $View$ denote the entire view of A at the conclusion of the above game. Then, $\mathsf{SD}(\langle \mathbf{r}, View \rangle, \langle \mathcal{U}_l, View \rangle) \leq \delta$. $SD()$ is again the statistical difference as defined in [2] and $\mathcal{U}_l$ the uniform distribution over l-bit strings.
- **(Error-tolerance)** For all $\mathbf{w}'$ with $\mathsf{d}(\mathbf{w}_0, \mathbf{w}') \leq t$, we have $\mathsf{Rec}(\mathbf{w}', \mathbf{pub}) = \mathbf{r}$.

2.3 Robust fuzzy extractor constructions

After we have defined the various constructions we will now take a look at how they can be realized. In this paper we only look at constructions for the Hamming distance metric, since this is used in PUF-based key generation. The main building block of all constructions is the secure sketch. Two secure sketch constructions for the Hamming distance metric were proposed by Dodis *et al.* [10], the code-offset and the syndrome construction.

2.3.1 Secure sketches for the Hamming distance metric

The code-offset construction [10] is based on the fuzzy commitment proposed by Juels and Wattenberg [13] and is defined as follows:

Definition 6: Code-offset construction: During sketching, choose a random codeword $\mathbf{x} \in \mathbf{C}$ and compute the helper data $\mathbf{s} = \mathsf{SS}(\mathbf{w}, \mathbf{x}) = \mathbf{w} \oplus \mathbf{x}$. For recovery, compute $\bar{\mathbf{x}} = \mathsf{decode}(\mathbf{w}' \oplus \mathbf{s})$, where $\mathsf{decode}()$ denotes the decoding procedure of the error-correction code $\mathbf{C}$ and $\mathbf{w}'$ the potentially noisy PUF response. Then $\bar{\mathbf{w}} = \mathsf{Rec}(\mathbf{w}', \mathbf{s}) = \mathbf{s} \oplus \bar{\mathbf{x}}$ with $\bar{\mathbf{w}} == \mathbf{w}$ if $\mathsf{d}(\mathbf{w}', \mathbf{w}) \leq t$.

The syndrome construction requires the code $\mathbf{C}$ to be a linear binary error-correction code and works as follows:

Definition 7: Syndrome construction: During sketching, compute $\mathbf{s} = \mathsf{SS}(\mathbf{w}, \mathbf{x}) = \mathbf{w} \cdot \mathbf{H^T}$, where $\mathbf{H^T}$ is the transposed parity-check-matrix of the used linear error-correction code $\mathbf{C}$. For recovery, compute $\mathbf{s}' = \mathbf{w}' \cdot \mathbf{H^T}$. Determine $\bar{\mathbf{e}} = \mathsf{locate}(\mathbf{s}' \oplus \mathbf{s})$ by using the error-location algorithm $\mathsf{locate}()$ of the code $\mathbf{C}$. Then $\bar{\mathbf{w}} = \mathsf{Rec}(\mathbf{w}', \mathbf{s}) = \mathbf{w}' \oplus \bar{\mathbf{e}}$ with $\bar{\mathbf{w}} = \mathbf{w}$ if $\mathsf{d}(\mathbf{w}', \mathbf{w}) \leq t$.

Both the code-offset and the syndrome construction are popular for PUF-based key generation.

2.3.2 Hash-based robust fuzzy extractor

Boyen *et al.* showed how robust fuzzy extractors can be built using any well-formed secure sketch in conjunction with a hash function. This construction, which we will denote as *hash-based construction* in the remainder of the paper is provably secure in the random oracle model [2]. This hash-based construction works as follows:

Definition 8: Assume we are given two hash functions $\mathsf{H}_1, \mathsf{H}_2 : \{0,1\}^* \rightarrow \{0,1\}^l$ (in practice a single hash function can be used by prepending a padding) and a well-formed secure sketch $(\mathsf{SS}, \mathsf{Rec})$. The hash-based robust fuzzy extractor $(\mathsf{Ext}, \mathsf{Rec})$ with $(\mathbf{r}, \mathbf{pub}) = \mathsf{Ext}(\mathbf{w})$ and $\bar{\mathbf{r}} = \mathsf{Rec}(\mathbf{w}', \mathbf{pub})$ works as follows:[3]

- $(\mathbf{r}, \mathbf{pub}) = \mathsf{Ext}(\mathbf{w})$: Let $\mathbf{s} = \mathsf{SS}(\mathbf{w})$. Output $\mathbf{pub} = (\mathbf{s}, \mathsf{H}_1(\mathbf{w}, \mathbf{s}))$ and $\mathbf{r} = \mathsf{H}_2(\mathbf{w}, \mathbf{s})$.
- $\bar{\mathbf{r}} = \mathsf{Rec}(\mathbf{w}', \mathbf{pub})$: Parse $\mathbf{pub}$ as $(\mathbf{s}, \mathbf{h})$ and set $\bar{\mathbf{w}} = \mathsf{Rec}(\mathbf{w}', \mathbf{s})$. If $\mathsf{d}(\bar{\mathbf{w}}, \mathbf{w}') \leq t$ and $\bar{\mathbf{h}} == \mathsf{H}_1(\bar{\mathbf{w}}, \mathbf{s})$ then output $\bar{\mathbf{r}} = \mathsf{H}_2(\bar{\mathbf{w}})$ else output $\bar{\mathbf{r}} = \perp$.

How the hash-based robust fuzzy extractor works in conjunction with the code-offset construction is illustrated in Figure 1. One great advantage of this construction is that in this way any fuzzy extractor can be extended to a robust fuzzy extractor. Hence, when discussing fuzzy extractor constructions for PUF-based key generation, it is often noted that in case resistance against active attackers

3. It is also possible to compute r with $\bar{\mathbf{r}} = \mathsf{H}_2(\bar{\mathbf{w}}, \mathbf{s})$ as it was done in [11]. Yet the original proposal in [2] uses $\bar{\mathbf{r}} = \mathsf{H}_2(\bar{\mathbf{w}})$.

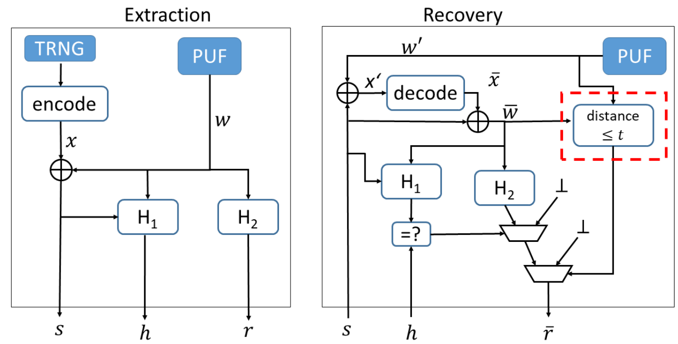


Fig. 1: Illustration of the robust fuzzy extractor when it is used in conjunction with the code-offset construct. On the left the extraction phase and on the right the recovery phase is depicted. The distance check $\mathsf{d}(\mathbf{w}', \mathbf{w}) \leq t$ highlighted in the red square is often overlooked in PUF papers. A fuzzy extractor without this distance check is called RFE-like construct in this paper and attacks against such a construct will be presented in Section 4.

is needed, the fuzzy extractor can easily be turned into a robust fuzzy extractor. However, we want to point to a detail that is often overlooked: In a fuzzy extractor $\mathsf{d}(\bar{\mathbf{w}}, \mathbf{w}') \leq t$ is only a correctness requirement, i.e., only guarantees that the errors are corrected. It has nothing to do with security and hence an engineer can simply ignore this part without jeopardizing security. Indeed, Dodis *et al.* [9] formalized this as *relaxed notion of correctness*, since better error correction rates can be achieved with such relaxed notations.

In a robust fuzzy extractor on the other hand $\mathsf{d}(\bar{\mathbf{w}}, \mathbf{w}') \leq t$ is a security requirement. If this inequality is not satisfied, the robust fuzzy extractor needs to return $\perp$. In other words, the robust fuzzy extractor requires a well-formed secure sketch not because of a correctness requirement, but because of a security requirement. Hence, this aspect cannot be ignored without security implications.

2.3.3 Fuzzy extractors for PUF-based key generation

In this Subsection we will briefly discuss some of the different fuzzy extractors that have been proposed in conjunction with PUF-based key generation. In earlier works BCH codes have been used [12], [24]. However, the superior error correction capability of concatenated codes was pointed out by Bösch *et al.* in 2008 [1]. In particular, they proposed a simple inner repetition code with an outer Reed–Muller or Golay code. Concatenated codes with BCH codes in conjunction with repetition codes have been used in the literature as well [20]. In 2009 the idea to use soft-decision error correction was first introduced by Maes *et al.* in conjunction with PUF-based error correction [18], [19]. The idea is to collect additional information about the reliability of each PUF response bit and add this as additional helper data during the generation phase. During reconstruction, soft-decision error correction codes can use this information to considerably decrease the probability of a decoding error. For this, concatenated codes with soft-decision repetition and Reed–Muller codes were proposed. One disadvantage of this approach is that the soft-decision information needs to be collected first, which might not always be trivial. How to overcome this problem was presented in 2012 by van der Leest *et al.* [16], by using a hard-in soft-out inner code (repetition decoder) in conjunction with a soft-decision outer code (Reed–Muller or Golay code). Compared to hard-decision decoding, considerably better error correction rates are achieved without the need to collect additional information during the generation phase. The same hard-in soft-out decoding was also used in [17]. Similarly, the generalized concatenated code constructions by Puchinger *et al.* [21] also use hard-in soft-out decoding of Reed–Muller codes.

3 Impossibility results for BCH codes

In order to show the feasibility of fuzzy extractors, Dodis *et al.* showed that BCH codes can be used to construct fuzzy extractors based on the code-offset and syndrome construction [10]. Later work showed how every fuzzy extractor can be turned into a robust fuzzy extractor [2] and, hence, at least in theory BCH codes can also be used to build robust fuzzy extractors. In this section we look at the problem from a more practical perspective, by investigating if the parameters of the security proof can also be met for realistic error-correction rates, as they are needed for PUF-based key generation. A typical goal in PUF-based key generation is to achieve a failure rate of less than 10^{-6} or 10^{-9} during key generation, with an assumed worst-case PUF reliability of around 85%[4]. This is assumed in, e.g., [1], [17], [18], [20], [21]. Note that in all of these papers concatenated code constructions are used that yield much better average error correction rates than non-concatenated code. However, concatenated codes have a significantly worse minimum error-correction capacity t (see Section 3.3), making them unsuitable for robust fuzzy extractors that follow the security proof from [2]. We therefore want to evaluate in this Section, if these results can also be achieved with a single BCH code, as it was proposed for robust fuzzy extractors by Boyen *et al.* [2].

3.1 Security bound for Boyen *et al.*'s hash-based robust fuzzy extractor construction

The robust fuzzy extractor proof from [2] provides a formula to compute the success probability ϵ of an active attacker. This equation allows the derivation of a security bound that the binary $[n, k, 2t + 1]$ BCH codes need to fulfill in order to be provably secure when used in robust fuzzy extractors.

4. Worst case reliability of 85% in this case means that the chance that a response bit flips is 15% in average for the worst environmental conditions that is still within specification (e.g. highest specified temperature).

Theorem 1: Security Bound:
A binary linear $[n, k, 2t + 1]$ code used in a robust sketch as defined in [2] that does not fulfill the following bound does not fulfill the corresponding security proof of Boyen *et al.:*

$$k > 1 + log_2(n) + log_2\left(\sum_{i=0}^{t} \binom{n}{i}\right) \tag{1}$$

Proof: From [2] Theorem 1 the success probability ϵ of an attacker is defined with:[5]

$$\epsilon = (4q_H + 2n \cdot v) \cdot 2^{-m'} \tag{2}$$

where q_H denotes the number of times an attacker is allowed to query the robust fuzzy extractor, v denotes the volume of the error correction code, and m' the remaining min-entropy after revealing the helper data. For binary linear $[n, k, 2t+1]$ codes v can be expressed as (see, e.g., [7], Sec. III.e.):

$$v = \sum_{i=0}^{t} \binom{n}{i} \tag{3}$$

For a secure sketch based on a $[n, k, 2t+1]$ code, the upper bound of the min-entropy m' can be written as $m' \leq n - (n - k) = k$. Note that we are looking for an impossibility result, i.e., we want to show that if the inequality does not hold the construction does not fulfill the security proof from [2]. We do not want to show that if the inequality holds that the constriction is secure. Therefore, we derive a lower bound of the attacker's success probability ϵ as follows:

$$\begin{aligned}
\epsilon &= (4q_H + 2n \cdot v) \cdot 2^{-m'} \\
\epsilon &\geq 2n \cdot \sum_{i=0}^{t} \binom{n}{i} \cdot 2^{-k} \\
log_2(\epsilon) &\geq log_2\left(2n \cdot \sum_{i=0}^{t} \binom{n}{i} \cdot 2^{-k}\right) \\
log_2(\epsilon) &\geq 1 + log_2(n) + log_2\left(\sum_{i=0}^{t} \binom{n}{i}\right) - k
\end{aligned} \tag{4}$$

Since the success probability ϵ needs to be smaller than 1, i.e., $\epsilon < 1$, one can simply define a lower bound as:

$$\begin{aligned}
log_2(1) > log_2(\epsilon) &\geq 1 + log_2(n) + log_2\left(\sum_{i=0}^{t} \binom{n}{i}\right) - k \\
0 &> 1 + log_2(n) + log_2\left(\sum_{i=0}^{t} \binom{n}{i}\right) - k \\
k &> 1 + log_2(n) + log_2\left(\sum_{i=0}^{t} \binom{n}{i}\right)
\end{aligned} \tag{5}$$

□

There are parameter choices for BCH codes that fulfill the bound. However, in practice, not only the security bound has to hold, but the resulting BCH code also needs to

5. We use the simplified lower bound of Theorem 1 [2] for the case that the hash output length was chosen appropriately.

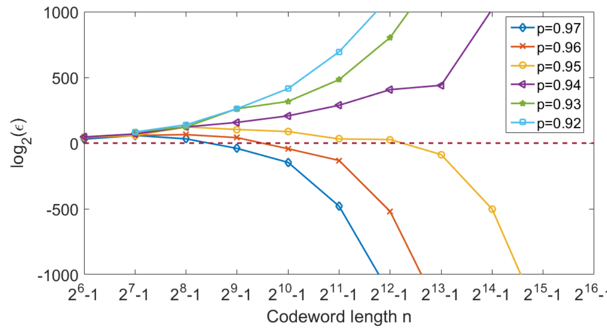
n	63	127	255	511	1023	2047	4095	8191	16383	32767	65535
p =0.92	-	27	45	85	126	231	413	819	1483	2861	5579
p =0.93	-	27	42	85	115	205	367	686	1321	2517	4902
p =0.94	15	23	42	59	102	179	330	597	1170	2340	4681
p =0.95	15	21	42	53	87	153	292	506	955	1829	3545
p =0.96	13	21	29	45	74	127	227	415	793	1483	2863
p =0.97	11	21	25	37	60	101	178	325	599	1133	2184

TABLE 2: Lower bound of the required number of correctable errors t for different codeword sizes n and reliabilities p to achieve a failure rate of less than 10^{-6}.

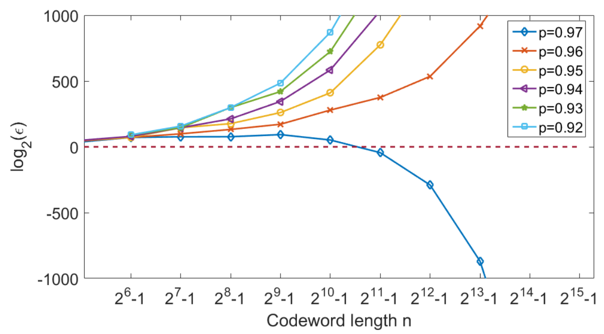
achieve the required error correction rate. In particular, usually the probability of a key generation failure P_{fail} in a worst-case scenario should be below a defined threshold. A simple but widely used error-model is that each PUF response bit flips independently with the same probability $1-p$ (i.e., a "naive, homogeneous" noise model [3]). Typical parameters found in the literature are $p \approx 0.85$ and P_{fail} around 10^{-6} to 10^{-9}. Table 2 shows the minimum value for t for $[n, k, 2t+1]$ BCH codes and different codeword sizes n and reliability values p to achieve a failure rate of less than $P_{fail} < 10^{-6}$ using the naive, homogenous noise model[6]. In a way, the values in the table are an "error-tolerance bound", since only error-correcting codes that can correct at least that many bit errors are reliable enough for a robust fuzzy extractor.

Since we now have a security bound as well as a lower bound on the required error correction capacity, we can verify whether or not building a secure robust fuzzy extractor based on BCH codes is possible. For this purpose we tested the security bound by using the BCH parameters derived in Table 2 and computing the corresponding ϵ value. In particular, we calculated the bounded $log_2(\epsilon)$ according to Equation (4) using Sage for the $[n, k, 2t+1]$ BCH codes that were closest to the values listed in Table 2. The result can be found in Figure 2a. As one can see, only for $p > 0.94$ the value is negative, i.e., fulfills the security bound. For all other values with $p \leq 0.94\%$ the security proof provided in [2] does not hold.

Note that for PUF-based key generation much lower reliabilities than 94% are needed (e.g. $p = 85\%$) and, hence, it is currently not possible to build a robust fuzzy extractor with BCH codes for PUF-based key generation, that is provably secure according to the proof provided by Boyen *et al.* [2].



(a) hash-based construction



(b) standard model construction

Fig. 2: The upper bound of the success probability ϵ of an attacker for a helper data manipulation attack against a) the hash-based construction from [2] and b) the standard-model construction from [8]. For each codeword size n and PUF reliability value p the best BCH parameters were chosen to compute ϵ. Note that a positive value means that for the corresponding reliability and codeword sizes the construction does not offer provable security against helper data manipulation attacks.

3.2 Security bound for Dodis *et al.*'s robust fuzzy extractor construction

The robust fuzzy extractor based on hash functions proposed by Boyen *et al.* [2] is provably secure in the random oracle model but not the general model. Therefore, a different construction was proposed by Dodis *et al.* that is also provably secure in the general model [8]. However, this provable security in the general model comes at the cost of a significantly reduced efficiency in terms of security parameters. In particular, it looses half of the min-entropy m'. Therefore the security bound for this construction is reduced to:

$$\frac{k}{2} \geq log\left(\sum_{i=0}^{t}\binom{n}{i}\right) + log\left(2\left\lceil\frac{k}{n-k}+2\right\rceil\right) - \frac{1}{2} \qquad (6)$$

Details of how this bound is derived can be found in the Appendix. We again computed the ϵ value for the BCH parameters provided in Table 2. The results can be found in Figure 2b. In this case, even for reliability values of $p = 98\%$ no BCH code exists that fulfills the security proof

6. The advantage of the naive, homogenous noise model is that it is independent of the PUF technology and provides the average failure rate over all devices. Note that it does not incorporate burst-errors or gives device-specific failure rates. The computation was done based on the CDF of a binomial distribution.

of Dodis *et al.* [8].

3.3 Impact of concatenated codes on the security bound

Before looking into concatenated codes, let us first consider the case that a message $\mathbf{m}$ is split into l blocks of size n and each block is decoded by a BCH code (or other linear code). Assume that a BCH code is used of which the maximum as well as the minimum number of correctable errors within any given code word is t. If we concatenate l codeword blocks, the maximum number of correctable errors is $t \cdot l$, while the minimum number of correctable errors is still only t, since $t+1$ errors in a single block will result in a decoding failure. Hence, for more than t errors the robust fuzzy extractor needs to output $\perp$, which makes it completely impractical. Otherwise, such a construction would not be in line with the security proofs of Boyen *et al.* [2] and Dodis *et al.* [8].

A crucial error correction concept in practice is the use of concatenated codes. Nearly all PUF-based key generation proposals use concatenated codes, which first encode the message with an inner code and then again with an outer code. In many cases the inner codes is a simple repetition code, while the outer code is more complex. The main idea is that the average error correction capability is considerably higher, even though the minimum number of correctable errors is much smaller than using a single large code. In practice, this average number of correctable errors is what really matters to determine the failure probability P_{fail} for PUF-based key generation. But since t is defined by the *minimum* number and not the average number of correctable errors, using concatenated codes in a well-formed secure sketch is impossible. For example, in [20] a [7,1,7] repetition code, concatenated with a [318,174,35] BCH code was used. The repetition code can correct 3 errors in a 7-bit codeword, and the BCH code can correct up to 17 errors in a 318-bit codeword. The concatenated code has a codeword length of $n = 318 \cdot 7 = 2,226$. The minimum number of bit errors for a decoding error to occur is $18 \cdot 4 = 72$. On the other hand, the maximum number of errors such that a codeword can still be decoded correctly is $17 \cdot 7 + 301 \cdot 3 = 1,022$. When used in a well-formed secure sketch as required for the robust fuzzy extractor constructions from [2] and [8], the number of correctable errors would have to be set to t=72. Hence, such a construction would be impractical and considerably worse than a non-concatenated code.

From an engineering perspective, it is tempting to ignore the requirement of the proof that a well-formed secure sketch is needed, and e.g. set t to the average or highest number of correctable errors. While this violates the proof, such a construction might still be secure considering that no attack on such a construction is known. This scenario is discussed in greater detail in the next section.

4 Helper data manipulation attack

In this section the general attack strategy for the new helper data manipulation attacks is described, before some specific attacks against specific error correction strategies and implementations are introduced in the next Section.

4.1 Attack model

Recently, Delvaux *et al.* [3] introduced a helper data manipulation attack on a soft-decision error-correction strategy, in which modified helper data is sent to the PUF-enabled device and the attacker observed whether or not a decoding failure occurred. With each query the attacker learns information about the PUF response and can eventually reconstruct the entire response in a divide-and-conquer-like fashion.

In this paper we consider that instead of a fuzzy extractor an RFE-like construct is used. An RFE-like construction is similar to the hash-based construction by Boyen *et al.* [2], but neglects the distance check t (see Figure 1). The RFE-like construction is tempting to use, since it allows the use of concatenated codes. The RFE-like construct prevents helper data manipulation attacks as described above, since the key recovery always fails if the attacker does not transmit a valid hash value $\mathbf{h}$. And without knowing the reconstructed PUF response $\bar{\mathbf{w}}$, the attacker cannot compute a valid hash value $\mathbf{h}_a$ for a modified helper data $\mathbf{s}_a$.

The new attack strategy is therefore not to try to learn the original PUF response $\mathbf{w}$, but instead to try to set the reconstructed PUF response $\bar{\mathbf{w}}$ to a value known by the attacker. This, in turn, enables the attacker to compute a valid hash value $\mathbf{h}_a$. Note that this also means that the reconstructed key $\bar{\mathbf{r}}$ is known by the attacker, but is not the same as the original key. Hence, the attacker does not learn the original key $\mathbf{r}$ with such an attack. Whether or not this is a reasonable attack goal depends on the application the key is used for. For example, imaging that the PUF-derived key is used to encrypt an externally stored bitstream. In the helper data manipulation attack from [3], the attacker would have learned the encryption key of the bitstream. This would allow the attacker to both decrypt the original bitstream as well as supply the device with its own bitstream. In our helper data manipulation attack scenario on the other hand the attacker would be able to supply a different bitstream to the device but would not be able to decrypt the original bitstream.

Other scenarios in which setting the key is a legitimate attack goal is if the key is used for access control to the device. In this case it is sufficient for the attacker to set a new key without learning the old key as the main goal is to get access to the device. If on the other hand the goal is to create a software clone of a PUF device, e.g. because it is used as an anti-counterfeiting mechanisms, setting a key is not enough. In this case the original key is needed.

However, the presented helper data manipulation attacks can also be extended from only setting a key to additionally recovering the original key. When and how this can be achieved is briefly discussed in Section 4.3. But the focus of this paper is only to set the key to defeat the security goal of a robust fuzzy extractor. The attacker's capability can be summarized as follows:

Attacker		PUF device
Given: helper data $\mathbf{s}$, with $\mathbf{s} = \mathbf{x} \oplus \mathbf{w}$		Given: noisy PUF function, $\mathsf{PUF}() = \mathbf{w} \oplus \mathbf{e}$
Choose attack vector $\mathbf{e}_a$ such that $\bar{\mathbf{x}}_a = \mathsf{decode}(\mathbf{c}_i \oplus \mathbf{e}_a), \forall \mathbf{c}_i \in \mathbf{C_a}$ with $\mathbf{C_a} \subseteq \mathbf{C}$ and $\lvert\mathbf{C_a}\rvert$ maximized $\mathbf{s}_a = \mathbf{s} \oplus \mathbf{e}_a$ $\bar{\mathbf{w}}_a = \mathbf{s}_a \oplus \bar{\mathbf{x}}_a$ $\mathbf{h}_a = \mathsf{H}_1(\mathbf{s}_a, \bar{\mathbf{w}}_a)$ $\mathbf{r}_a = \mathsf{H}_2(\bar{\mathbf{w}}_a)$		
	$\xrightarrow{\mathbf{s}_a, \mathbf{h}_a}$	
		$\mathbf{w}' = \mathsf{PUF}() = \mathbf{w} \oplus \mathbf{e}$ $\mathbf{x}' = \mathbf{w}' \oplus \mathbf{s}_a$ $\quad = (\mathbf{w} \oplus \mathbf{e}) \oplus (\mathbf{w} \oplus \mathbf{x} \oplus \mathbf{e}_a) = \mathbf{x} \oplus \mathbf{e} \oplus \mathbf{e}_a$ $\bar{\mathbf{x}} = \mathsf{decode}(\mathbf{x}') = \mathsf{decode}(\mathbf{x} \oplus \mathbf{e} \oplus \mathbf{e}_a)$ $\bar{\mathbf{w}} = \mathbf{s}_a \oplus \bar{\mathbf{x}}$ $\bar{\mathbf{h}} = \mathsf{H}_1(\mathbf{s}_a, \bar{\mathbf{w}})$ IF $\bar{\mathbf{h}}$==$\mathbf{h}_a$ set $\bar{\mathbf{r}} = \mathsf{H}_2(\bar{\mathbf{w}})$ and status='ok' ELSE set $\bar{\mathbf{r}} = \perp$ and status='error'
	$\xleftarrow{\text{status}}$	
IF status=='error' repeat attack ELSE success and $\mathbf{r}_a == \bar{\mathbf{r}}$ and $\bar{\mathbf{w}}_a == \bar{\mathbf{w}}$		

TABLE 3: General description of the new helper data manipulation attack on an RFE-like construct.

- Attacker's goal:
  - Make the PUF-enabled device reconstruct a key $\bar{\mathbf{r}}_\mathbf{a}$ that is known to the attacker (but which differs from the original key $\mathbf{r}$).
- Attacker's capabilities:
  - The attacker has read and write access to the helper data ($\mathbf{s}$ and $\mathbf{h}$)
  - The attacker can verify if the predicted $\bar{\mathbf{r}}_\mathbf{a}$ has been reconstructed by the PUF device

4.2 New attack strategy

The main idea behind the new helper data manipulation attack is to send modified helper data $\mathbf{s}_a$ to the PUF-enabled device such that during the recovery phase it decodes to a specific known codeword $\bar{\mathbf{x}}_a = \mathsf{decode}(\mathbf{w}' \oplus \mathbf{s}_a)$ with a very high probability, independent of the PUF response $\mathbf{w}'$. This, in turn, implies that the PUF device will recover a PUF response $\bar{\mathbf{w}}_a = \bar{\mathbf{s}}_a \oplus \bar{\mathbf{x}}_a$ which the attacker can predict with a high probability. The attacker can then compute a valid hash value $\mathbf{h}_a = \mathsf{H}_1(\mathbf{s}_a, \bar{\mathbf{w}}_a)$. If the attack succeeded, the PUF device and the attacker then share a common key $\bar{\mathbf{r}}_a = \mathsf{H}_2(\bar{\mathbf{w}}_a)$.

The entire attack is depicted in Table 3. The crucial point is, whether or not the attacker is able to modify the helper data $\mathbf{s}$ in a way that during the error-correction phase the PUF device will decode towards the attacker's prediction $\bar{\mathbf{x}_\mathbf{a}}$ with a high probability. Whether or not this is possible strongly depends on *i)* the used error-correction code and *ii)* the implementation of the error-correction decoding. In the next section we will show some concrete attacks on implementations of different error-correction codes that have been proposed in the context of PUF-based key storage.

4.3 Extending attack to extract original key

Depending on the applications and used error-correction codes, it might be possible to not only set the key but to also recover the key. In practice, often not a single long code is used for the codeword $\mathbf{x}$, but instead the the codeword $\mathbf{x}$ actually consists of l smaller codewords that are concatenated together. This is actually true for all the codes we consider in our attack section. In this case, an attacker can perform a helper data manipulation attack that tries to set the codeword $\bar{\mathbf{x}}$ for $l-1$ codeword blocks but leaves one untouched. For the helper data manipulation attack to succeed, the attacker then guesses the unmodified codeword and tests the guess using the appropriate $\bar{\mathbf{w}}_a$ to compute the hash $\mathbf{h}_a$. If the unmodified code word has k bit of entropy, the attacker has in average to guess 2^{k-1} times till he succeeds and can proceed to the next codeword block. This way the attacker could learn the entire original codeword $\mathbf{x}$, and, hence, also the original PUF response $\mathbf{w} = \mathbf{x} \oplus \mathbf{s}$ and key $\mathbf{r}$ at the costs of an increased attack complexity of $l \cdot 2^{k-1}$.

Note that this strategy also works, if only a fuzzy extractor (as opposed to an RFE-like fuzzy extractor) is used, as long as the attacker can verify if the recovered key is the one he assumed it is. This greatly increases the practical relevance of the attacks discussed in the next section.

5 Example attacks

In this Section several attacks on popular error-correction codes for PUF-based key generation are shown. In particular, several decoding strategies for Reed–Muller codes are attacked, as well as soft-decision decoding and even-numbered repetition decoding.

One of the most popular error-correction codes for PUF-based key generation are Reed–Muller codes, which are for example used in [1], [16], [18], [19], [21]. In the following, we will show that different implementation strategies for Reed–Muller codes exists that can be attacked using the new helper data manipulation attack. For a successful attack, we need to find an error pattern $\mathbf{e}_a$, such that the decoding algorithm decodes the codeword $\mathbf{x}' \oplus \mathbf{e}_a$ to the same codeword $\bar{\mathbf{x}}_a$ for most codewords $\mathbf{x}'$. There are several possible decoding strategies for Reed–Muller codes. We consider three popular decoding strategies: soft- and hard-decision maximum-likelihood decoding (SDML, ML), soft-decision Generalized Multiple Concatenated codes (GMC) decoding, and classic Reed decoding based on a majority logic vote.

5.1 Attacking SDML decoding

The SDML decoding procedure simply consists of generating all possible codewords and returning the one with the smallest Hamming distance to the received vector. The key observation for our attack on SDML decoding is that for some error vectors $\mathbf{e}_a$ several codewords have the same minimal Hamming distance. In this case always the first or last tested codeword will be chosen in most implementations[7].

5.1.1 The noise free case

To provide a concrete example, let us look at a [16,5,8] Reed–Muller code which contains $2^k = 2^5 = 32$ different 16-bit codewords $\mathbf{x}_i$. Let us denote the codeword $\mathbf{x}_0$ as the codeword consisting of all-zeros $\mathbf{x}_0 = [0, .., 0]$, and $\mathbf{x}_1 = [1, .., 1]$ the codeword consisting of all-ones. If we add the following error vector $\mathbf{e}_a = [0110101011000000]$ to $\mathbf{x}_i$ then:

$$\begin{aligned} \mathsf{HD}(\mathbf{x}_0, \mathbf{x}_i \oplus \mathbf{e}_a) = 6 \text{ or } \mathsf{HD}(\mathbf{x}_1, \mathbf{x}_i \oplus \mathbf{e}_a) = 6 \\ \text{and } \mathsf{HD}(\mathbf{x}_j, \mathbf{x}_i \oplus \mathbf{e}_a) = \{6, 10\} \ \forall \ i, j \geq 2 \end{aligned} \tag{7}$$

In this case, depending on the codeword $\mathbf{x_i}$, the maximum-likelihood decoding always decodes to either $\mathbf{x}_0$ or $\mathbf{x}_1$ for attack vector $\mathbf{e}_a$. In particular, there are 16 codewords $\mathbf{x_i}$ such $\mathbf{x}_0 = \mathsf{decode}(\mathbf{x}_i \oplus \mathbf{e}_a)$ and 16 codewords $\mathbf{x}_i$ such that $\mathbf{x}_1 = \mathsf{decode}(\mathbf{x}_i \oplus \mathbf{e}_a)$. Hence, by manipulating the helper data with error vector $\mathbf{e}_a$, only one bit of entropy remains from the original 5 bits of entropy in the noise-free case.

For their PUF design, van der Leest *et al.* [16] proposed to use 35 blocks of a concatenated code construction consisting of a hard-in soft-out [7,1,7] repetition code as an inner code and a [16,5,8] Reed–Muller code with SDML decoding as an outer code to derive a 175-bit secret. Let us first consider the noise free case. In order to defeat the RFE-like construction, the attacker has to correctly guess 35 blocks, i.e. predict 35 times the decoded codeword $\bar{\mathbf{x}}_\mathbf{a}$ correctly to be able to compute the correct hash $\bar{\mathbf{h}}_a$ and key $\bar{\mathbf{r}}_a$. After the helper data manipulation one bit of entropy is left per block and a resulting entropy and min-entropy of 35 bits all codeword blocks. One way to interpret the min-entropy is that the success probability for the most likely decoded codeword $\bar{\mathbf{x}}_a$ is 2^{-35}. But it should be noted, that testing a key requires the attacker to send a manipulated helper data to the PUF device. Even for an entropy of 35 bits the attack would be quite hard to execute in practice, since the PUF would have to be challenge around 2^{35} times. But please also note that a full bit-entropy of the PUF is assumed which in practice might be reduced, e.g., due to a bias of response bits.

7. It is also possible to choose a random codeword instead, in which case the attack would not work. However, this is typically more difficult to implement especially for hardware implementation.

reliability	min-entropy (per codeword)	entropy (per codeword)	min-entropy (per 175 bits)
soft-decision			
100%	1.0	1.0	35.0
99%	2.0	3.6	69.6
98%	2.8	4.5	98.2
97%	3.4	4.8	118.5
96%	3.8	4.9	130.8
95%	4.0	4.9	138.8
hard-decision			
100%	1.0	1.0	35.0
95%	1.0	1.0	35.0
90%	1.0	1.2	36.4
85%	1.2	1.8	41.1
80%	1.5	2.7	51.8
Without attack			
–	5	5	175

TABLE 4: Top: Overview of helper data manipulation attacks on a hard-in soft-out [7,1,7] repetition code in conjunction with a [16,5,8] Reed–Muller code using soft-decision SDML decoding. Bottom: hard-decision ML decoding of a concatenated [7,1,7] repetition code with a [16,5,8] Reed–Muller code

5.1.2 Impact of noise on the attack

In practice the PUF response $\mathbf{w}'$ will be noisy and hence a legitimate question is how well the attack works in the presence of noise. To evaluate this question, we performed following analysis. In a first step, 100,000 random binary response strings are generated in MATLAB and the corresponding helper data for the code-offset construction and the error correction code are computed. Then each response is flipped with probability $1 - p$ to simulate noise (i.e. we use the naive homogeneous noise model in which each response bit has the same failure probability). The helper data is manipulated according to the attack strategy. The noisy responses and the modified helper data are passed to the Rec algorithm and the decoded responses are stored. Based on this analysis, the probability that a decoding to a certain response $\bar{\mathbf{w}}_i$ occurs is computed, which in turn allows us to compute the min-entropy and entropy of the recovered response $\bar{\mathbf{w}}$ in the presence of the attack. One thing to consider is that to estimate the required error-correction capacity of a code the worst-case PUF reliability under environmental conditions is used. But during an attack the environmental conditions can be kept constant to reduce the noise to a minimum. Therefore, the reliability during an attack is considerably higher than the worst-case reliability used to select the code. For SRAM PUFs for example, reliability values around 96-98% are not uncommon when the environmental conditions are kept constant while their worst case reliably can be easily 10% worse [23].

The top of Table 4 shows the results for the attack on the SDML construction from [16] while the bottom shows the same attack for hard-decision maximum likelihood decoding as proposed in [1]. The attack on soft-decision decoding quickly becomes very difficult to perform in the presence of noise. Even with a reliability of 99%, the min-entropy within a 175-bit block is only reduced to 69.6. However, when hard-decision maximum likelihood decoding is used, the attack is very resistant to noise. This is due to the fact that the inner repetition code corrects most of the noise. In this case the min-entropy of a 175-bit block is only slightly increased from 35 to 36.4 for a reduction of the PUF reliability to 90%. This shows that helper data manipulation attacks are not restricted to soft-decision decoding. Hard-decision decoding can actually be easier to attack in some cases.

5.2 Attacking GMC decoding

Another popular decoding algorithm for Reed–Muller codes is the Generalized Multiple Concatenated code (GMC) decoding [22]. It is, e.g., used in [19] and [21] in conjunction with PUF-based key generation. The main idea of this soft-decision decoding algorithm is to treat the Reed–Muller code as a concatenation of multiple smaller codes, and recursively decode these codes. The smallest code that is decoded is simply a repetition code, which has a very low decoding complexity. The algorithm can be used as a soft-decision decoder, and with some slight modifications also supports erasures. We implemented the GMC decoding in MATLAB as a target implementation. Again, the [16,5,8] Reed–Muller code was used and an error pattern for a helper data manipulation attack was found by simply testing all 2^{16} possible error pattern. It turned out that the helper data manipulation attack on our GMC decoding implementation is actually more powerful than the one on SDML decoding.

For the noise-free case, our GMC implementation always decodes to the same codeword when supplied with following error vector $\mathbf{e}_a = [0000001011001010]$, i.e.:

$$\mathbf{x}_0 = \mathsf{decode}(\mathbf{x}_i \oplus \mathbf{e}_a)\ \forall \mathbf{x}_i \in \mathbf{C} \tag{8}$$

Hence, the resulting entropy is zero, since always $\mathbf{x}_0$ is decoded. The attack complexity for a [7,1,7] hard-in soft-out repetition code as an inner code in conjunction with the tested soft-decision GMC $[16, 5, 8]$ Reed–Muller implementation for different error rates is summarized in Table 5. For a noise level of 98% the attack complexity is quite reasonable and, hence, the attack can be considered practical. However, for larger noise levels the attack complexity increases significantly.

reliability	min-entropy (per codeword)	entropy (per codeword)	min-entropy (per 175 bits)
100%	0	0	0
99%	0.4	2.2	14.4
98%	0.9	3.7	31.3
97%	1.4	4.5	49.2
96%	1.9	5.0	66.6
95%	2.4	5.2	82.7
Without attack			
–	5	5	175

TABLE 5: Overview of helper data manipulation attacks on a hard-in soft-out [7,1,7] repetition code in conjunction with a [16,5,8] Reed–Muller code using soft-decision GMC decoding.

reliability	min-entropy (per codeword)	entropy (per codeword)	min-entropy (per 175 bits)
100%	0.2	0.5	6.8
95%	0.2	0.6	6.8
90%	0.2	0.7	7.4
85%	0.3	1.3	11.0
Without attack			
–	5	5	175

TABLE 6: Results of helper data manipulation attacks on a hard-decision $[7, 1, 7]$ repetition code in conjunction with a $[16, 5, 8]$ Reed–Muller code using classic majority decoding.

5.3 Attacking Reed decoding

Another decoding algorithm considered in this paper is the classic hard-decision Reed decoding based on majority logic. For this purpose we took a publicly available MATLAB implementation of a Reed–Muller majority logic decoding and again searched for an error vector suitable for a helper data manipulation attack. The attack vector we found was actually better than for the SDML decoding but slightly worse than GMC decoding. For error vector $\mathbf{e}_a = [1001011000000000]$ the implementation we used decoded to the all-zero codeword $\mathbf{x}_0$ for 28 of the 32 codewords $\mathbf{x}_i$. The four codewords that did not decode to $\mathbf{x}_0$ decoded to the bit vector consisting of all zeros besides the least significant bit which was a one. Hence, in the noise-free case, a min-entropy of $28/32 = 0.2$ is achieved per block. However, when considering noise the helper data manipulation attack on classic hard-decision Reed decoding actually exhibits the best performance amongst all considered concatenated codes. This is once again due to the fact that the inner repetition code corrects most of the noise. If we use a [7,1,7] repetition code in conjunction with a $[16, 5, 8]$ Reed–Muller code even with a PUF reliability of only 85% the attack complexity is in the order of 2^{11}. The results of our attack on the concatenated construction can be found in Table 6.

For comparison, Figure 3 shows the development of the min-entropy of this attack when only a single Reed–Muller code is being used. Without the inner repetition coding, the attack would be considerably more difficult in a noisy environment.

5.4 Attacks on soft-decision and even-numbered repetition codes

Maes *et al.* were the first to propose soft-decision decoding for PUFs in 2009 [18]. They proposed to derive a reliability value for each PUF bit by querying the PUF multiple times

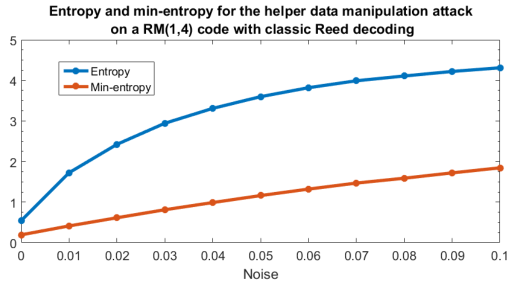


Fig. 3: The resulting entropy and min-entropy for a helper data manipulation attack on a [16,5,8] Reed–Muller code with classic Reed decoding based on majority logic. Note that the information content of a $[16, 5, 8]$ Reed–Muller code is 5 bits, i.e., without the attack the entropy and min-entropy would be 5.

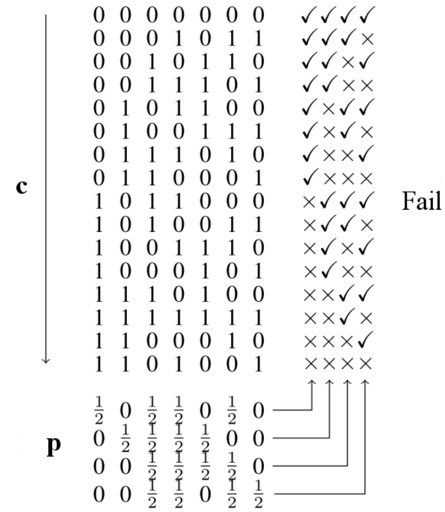


Fig. 4: Soft-decision manipulation attack for a [7,4,3] code with SDML decoding, taken from [3]. It is assumed that the first codeword is selected in case of a likelihood tie.

during the setup phase. This reliability vector $\mathbf{p}$ is then stored and provided as additional helper data to a soft-decision error correction code. Hence, in their scheme a second helper data string $\mathbf{p}$ is provided to the PUF device in addition to the helper data $\mathbf{s}$.

Such a soft-decision error-correction code was used by Delvaux *et al.* [3] to demonstrate their helper data manipulation attack on a fuzzy extractor-like construct(the soft-decision helper data algorithm [18] is not a fuzzy extractor as no bound on the min-entropy loss is provided). The attack is based on the fact that, if the probability $\mathbf{p}$ is set to exactly 0.5, then the corresponding bit will essentially be ignored during the soft-decision decoding. The authors showed a divide-and-conquer manipulation attack using SDML decoding of a simple $[7, 4, 3]$ BCH code as an example. The idea is to set $p_i = 0.5$ for some bits and observe if a decoding failure occurs. With each error pattern, the attacker recovers information about the codeword, until only one possible codeword remains. This attack idea is illustrated in Figure 4 [3].

While the divide-and-conquer approach is very efficient for fuzzy extractors, the RFE-like construction prevents this type of helper data manipulation attack, as it will always return a decoding failure unless a valid hash $\bar{\mathbf{h}}_a$ is supplied. However, applying the new attack strategy is straightforward for such a soft-decision decoding algorithm. As also observed in [3], by setting all values to $\mathbf{p_a} = (0.5, .., 0.5)$ in repetition codes, the codeword corresponding to 1 is equally likely than the codeword corresponding to 0. In this case, the decoder will decode always to either 1 or 0 depending on the implementation[8]. Hence, by setting $\mathbf{p_a} = (0.5, .., 0.5)$ for all codeword bits, the response bits are all decoded to either 0 or 1.

Of great practical relevance is also that the same attack works on even-numbered repetition codes, which have been proposed as a hard-in soft out outer code e.g. in [16]. The principle of a hard-in soft-out repetition code is fairly simple. The output is the Hamming weight of the codeword divided by the length of the codeword. The resulting value is basically the probability that the corresponding message bit is a 1. Three different constructions have been proposed in [16] based on repetition codes in conjunction with either Reed–Muller or Golay codes. Two of the proposals use an even numbered repetition code, including a $[8, 1, 8]$ hard-in soft-out repetition code with a $[24, 12, 8]$ soft-decision Golay code. If we flip half of the bits in an [8,1,8] repetition code, then a 0 and a 1 are equally likely and we basically get $p_i = 0.5$. In this case, the corresponding bit is decoded to either always 0 or always 1 in most implementation for the noise free case. Hence, by flipping half of the bits of the helper data the attacker forces the decoded codeword $\bar{\mathbf{x}}_a$ to be the all zero codeword. Note that for an uneven numbered repetition code, such a helper data manipulation attack is not possible.

5.4.1 *Impact of noise*

Of course, in the presence of noise the decoder might decode to a different codeword depending on which bits are noisy. To get a feeling how well such an attack would work in practice, we simulated the attack for several error rates and the [8,1,8] repetition code in conjunction with a [24,12,8] Golay code and a soft-decision Hackett decoder. In [16], 11 blocks of the concatenated code construction are used to generate a key with an entropy of 132 bit. Table 7 shows the result of the noise analysis based on the same experimental setup as discussed before.

As one can see, if the PUF reliability is very high, e.g. 99%, the attack is very practical with a min-entropy per codeword of 1.2 and a min-entropy for 132 bits of 12.9. One way to interpret the min-entropy for 132 bits is that the success probability of a helper data manipulation attack is $2^{-12.9}$ when the attacker chooses the most likely codeword to compute h_1. Hence, the min-entropy can be viewed

8. Note that just like the previous attacks this attack strongly depends on the implementations.

reliability	min-entropy (per codeword)	entropy (per codeword)	min-entropy (per 132 bits)
100%	0	0	0
99%	1.2	3.8	12.9
98%	2.0	6.1	22.4
97%	2.8	7.6	30.5
96%	3.4	8.7	37.9
95%	4.1	9.4	44.9
Without attack			
–	12	12	132

TABLE 7: Result of a helper data manipulation attack on a [8,1,8] hard-in soft-out repetition code as an outer code and a [24,12,8] Golay code with a soft-decision Hackett decoder for different noise levels.

as the attack complexity at least in case that the noise is identically and independently distributed (which is not necessarily true in practice, see [3]). For smaller reliability values such as 95% the attacks become quite difficult to perform in practice with a min-entropy of roughly 44.9.

6 Discussion

So far we have mainly presented negative results, i.e., showed that it is currently not possible to build a robust fuzzy extractor that fulfills the security proof, and that several practical error correction constructions are attackable. However, by no means does that mean that building a provably secure fuzzy extractor is in general impossible. What we need are new proofs and constructs.

6.1 Outlook: Secure decoding strategies

Looking forward, what is needed is a provably secure robust fuzzy extractor that works with efficient concatenated code constructions to achieve the required error correction rates. As a starting point, the hash-based construction seems to be very promising, since it has considerably better performance than the construction for the general model proposed in [8]. The helper data manipulation attacks discussed in Section 4 show that these attacks strongly depend on the used error correction code, the decoding strategy, as well as its implementation. But by no means do our results show that any error correction code or any implementation could be attacked using helper data manipulation attacks! In particular, some error correction codes have a very interesting property that makes them secure against the helper data manipulation attacks from Section 4.

Recall the *syndrome decoding* strategy from Definition 7, where during decoding in the first step an error polynomial $\bar{\mathbf{e}}$ is computed using the locate function, and in the second step this error polynomial is XORed with the received codeword $\mathbf{x}'$. What is important for us is that there are decoding strategies for BCH codes based on syndrome decoding for which the following equation holds for all codewords $\mathbf{x}_i$ of the $[n,k,2t+1]$ code $\mathbf{C}$:

$$\begin{gathered}\bar{\mathbf{e}}_j = \mathsf{locate}\big((\mathbf{x}_i \oplus \mathbf{e}_j)\cdot \mathbf{H}^T\big)\\ \forall\ \mathbf{x}_i \in \mathbf{C},\ \forall\ \mathbf{e}_j \in \{0,1\}^n \text{ and } \bar{\mathbf{e}}_j \in \{0,1\}^n\end{gathered} \tag{9}$$

In other words, the error polynomial $\bar{\mathbf{e}}_j$ is independent of the codeword $\mathbf{x}_i$ and only depends on the error polynomial $\mathbf{e}_j$. While it is possible to flip specific bits of the decoded codeword $\bar{\mathbf{x}}$ with a helper data manipulation attack, it is not possible to *set* specific bits. The attacker can no longer predict $\bar{\mathbf{x}}$ with an increased probability and can therefore also not compute a hash value $\bar{\mathbf{h}}_a$ which the PUF device will accept as valid. Hence, the helper data manipulation attacks presented in this paper do not work any longer, if a concatenated code construction is used in the RFE-like construction for which Equation (9) holds.

It therefore seems, that it should be possible to build secure robust fuzzy extractors, if the used error correction codes fulfills Equation (9). However, note that we only presented a security argument. Proving this in a more formal setting would be very interesting, since this would enable us to use a relaxed notion of correctness for robust fuzzy extractors, similar as it has been proposed for fuzzy extractors in [9]. This would allow the use of efficient concatenated code constructions and possibly even soft-decision decoding. But it is important to note that this way not a specific error-correction code construct can be shown to be provably secure. Instead, only specific error correction strategies could be proven to be secure. In other words, using a BCH code does not guarantee that the implemented decoding strategy indeed fulfills Equation (9). To give an example, consider an $[n, k, 2t+1]$ BCH code with a small k. The typical decoding strategy for a BCH code is syndrome decoding. But for codes with a small k, maximum-likelihood decoding can be used as well and might be very efficient, especially in hardware implementations. However, for maximum-likelihood decoding Equation (9) does not hold and helper data manipulation attacks are possible (see the attack in Section 5.1).

Another important, but difficult, aspect is the remaining min-entropy for concatenated codes if the PUF response $\mathbf{w}$ does not have full bit entropy. While there is a clear upper bound on the entropy loss in fuzzy extractors [10], using this bound makes building fuzzy extractors very challenging [15]. In [17], [4] a more in-depth discussion regarding this aspect is presented, including a considerably tighter bound than the one assumed in [15]. In how far the presented attacks can be improved by incorporating such reductions in the PUF response entropy is an interesting research question.

6.2 Random oracle model vs general model

From a practical perspective, the first robust fuzzy extractor proposal by Boyen *et al.* [2] based on hash functions is very compelling, since it only requires a hash function which needs to be implemented for the fuzzy extractor anyway. Furthermore, it is very straightforward and easy to implement and to understand. The fact that it is "only" secure in the random oracle model is not seen as a big problem from a practical perspective. To put it bluntly, hardware security engineers have much bigger problems than the assumptions in the random oracle model. For example, in practice it is typically impossible to determine the exact entropy within a PUF or a biometric reading. The best we can usually do is perform measurements and simulations and approximate the entropy based on some assumptions. However, this will never be hundred percent accurate and proving that the made assumptions are correct extremely challenging, if not impossible, unless very loose bounds are used. Therefore sacrificing nearly half of the entropy so that the system is also provably secure in the standard model does not really make sense from a practical perspective. As a result, the construction by Dodis *et al.* [8] that is provably secure in the general model has basically been ignored by the PUF community. It should also be noted that the problem of robust fuzzy extractors in general has been largely neglected by the PUF community. Usually, the need of security against an active attacker is acknowledged, but this is typically only followed by the remark that in this case hash-based robust fuzzy extractor construction should be used. However, it appears that the details of the proofs and definitions of the robust fuzzy extractor have not been really considered. Therefore, the fact that it is actually not possible to extend the popular fuzzy extractor constructions to robust fuzzy extractors has not been discussed.

From a theoretical perspective, the construction that is secure in the general model is more compelling since it has less assumptions. The construction for the standard model therefore has gained much more attention in theory oriented papers [14], [7]. However, the fact that the small error correction rate of robust fuzzy extractors is actually the most limiting factor for a provable, as well as practical, robust fuzzy extractor has not been identified. Coming up with a provably secure robust fuzzy extractor with a relaxed notion of correctness (i.e., based on a non well-formed secure sketch) would have a big practical impact, even if it is proven in a weaker security model than the random oracle model or general model.

7 Conclusion

This work shows that *i)* currently no robust fuzzy extractor construction exists that fulfills the security proof of Boyen *et al.* [2] while also achieving the required error correction rates for PUF based key generation, and *ii)* many implementations of decoding algorithms for error correction codes used in PUF-based key generation schemes are susceptible to helper data manipulation attacks, even when they are used in a fuzzy extractor setting with an additional hash value check against modifications. Please note that these attacks only work when the distance check t that is required in the construction of Boyen *et al.* is ignored. Hence, these attacks do not invalidate the results of Boyen *et al.* , but instead highlight that simply ignoring the check, as it seems to be the suggested in many practice oriented papers, is not a valid solution.

Hence, our results show that the problem of building robust fuzzy extractors is actually not solved yet. Our attack on the widely used Reed–Muller decoding shows that there is a great need to build robust fuzzy extractors, that are secure against helper data manipulation attacks. While this paper mainly presented negative results and attacks, this does not mean that building secure robust fuzzy extractors is a lost cause: By considering specific decoding strategies it seems that it should be possible to build both (provably) secure and practical robust fuzzy extractor. However, for this a combined effort of both practitioners, as well as theorist, is needed.

References

[1] Bösch, C., Guajardo, J., Sadeghi, A.R., Shokrollahi, J., Tuyls, P.: Efficient helper data key extractor on FPGAs. In: CHES 2008, LNCS, vol. 5154, pp. 181–197. Springer (2008)

[2] 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)

[3] Delvaux, J., Gu, D., Schellekens, D., Verbauwhede, I.: Helper data algorithms for PUF-based key generation: overview and analysis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 34(6), 889–902 (2015)

[4] Delvaux, J., Gu, D., Verbauwhede, I., Hiller, M., Yu, M.D.M.: Efficient fuzzy extraction of PUF-induced secrets: Theory and applications. In: CHES 2016. LNCS, Springer (2016)

[5] Delvaux, J., Verbauwhede, I.: Attacking PUF-based pattern matching key generators via helper data manipulation. In: Cryptographers Track at the RSA Conference (CT-RSA). pp. 106–131. Springer (2014)

[6] Delvaux, J., Verbauwhede, I.: Key-recovery attacks on various RO PUF constructions via helper data manipulation. In: Proceedings of the conference on Design, Automation & Test in Europe - DATE 2014. p. 72. European Design and Automation Association (2014)

[7] Dodis, Y., Kanukurthi, B., Katz, J., Reyzin, L., Smith, A.: Robust fuzzy extractors and authenticated key agreement from close secrets. IEEE Transactions on Information Theory 58(9), 6207–6222 (2012)

[8] Dodis, Y., Katz, J., Reyzin, L., Smith, A.: Robust fuzzy extractors and authenticated key agreement from close secrets. In: Annual International Cryptology Conference. pp. 232–250. Springer (2006)

[9] Dodis, Y., Ostrovsky, R., Reyzin, L., Smith, A.: Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. SIAM journal on computing 38(1), 97–139 (2008)

[10] Dodis, Y., Reyzin, L., Smith, A.: Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In: Eurocrypt 2004. LNCS, vol. 3027, pp. 523–540. Springer (2004)

[11] Dodis, Y.e.H.t.g.s.k.f.b., other noisy data, Reyzin, L., Smith, A.: Fuzzy Extractors, pp. 79–99. Springer London, London (2007), https://doi.org/10.1007/978-1-84628-984-2_5

[12] 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)

[13] Juels, A., Wattenberg, M.: A fuzzy commitment scheme. In: Proceedings of the 6th ACM conference on Computer and communications security. pp. 28–36. ACM (1999)

[14] Kanukurthi, B., Reyzin, L.: An improved robust fuzzy extractor. In: International Conference on Security and Cryptography for Networks. pp. 156–171. Springer (2008)

[15] Koeberl, P., Li, J., Rajan, A., Wu, W.: Entropy loss in PUF-based key generation schemes: The repetition code pitfall. In: HOST 2014. pp. 44–49. IEEE (2014)

[16] van der Leest, V., Preneel, B., van der Sluis, E.: Soft decision error correction for compact memory-based PUFs using a single enrollment. In: CHES 2012, LNCS, vol. 7428, pp. 268–282. Springer (2012)

[17] Maes, R., van der Leest, V., van der Sluis, E., Willems, F.: Secure key generation from biased PUFs. In: CHES 2015, pp. 517–534. LNCS, Springer (2015)

[18] Maes, R., Tuyls, P., Verbauwhede, I.: Low-overhead implementation of a soft decision helper data algorithm for SRAM PUFs. In: CHES 2009, LNCS, vol. 5747, pp. 332–347. Springer (2009)

[19] Maes, R., Tuyls, P., Verbauwhede, I.: A soft decision helper data algorithm for SRAM PUFs. In: IEEE International Symposium on Information Theory – ISIT 2009. pp. 2101–2105. IEEE (2009)

[20] Maes, R., Van Herrewege, A., Verbauwhede, I.: PUFKY: A fully functional PUF-based cryptographic key generator. In: CHES 2012, LNCS, vol. 7428, pp. 302–319. Springer (2012)

[21] Puchinger, S., Müelich, S., Bossert, M., Hiller, M., Sigl, G.: On error correction for physical unclonable functions. In: SCC 2015; 10th International ITG Conference on Systems, Communications and Coding; Proceedings of. pp. 1–6. VDE (2015)

[22] Schnabl, G., Bossert, M.: Soft-decision decoding of Reed-Muller codes as generalized multiple concatenated codes. IEEE Transactions on Information Theory 41(1), 304–308 (1995)

[23] Schrijen, G.J., van der Leest, V.: Comparative analysis of SRAM memories used as PUF primitives. In: Proceedings of the Conference on Design, Automation and Test in Europe. pp. 1319–1324. EDA Consortium (2012)

[24] Suh, G.E., O'Donnell, C.W., Sachdev, I., Devadas, S.: Design and implementation of the AEGIS single-chip secure processor using physical random functions. In: 32nd International Symposium on Computer Architecture (ISCA'05). pp. 25–36 (June 2005)

[25] Yu, M.M., Sowell, R., Singh, A., M'Raïhi, D., Devadas, S.: Performance metrics and empirical results of a PUF cryptographic key generation ASIC. In: HOST 2012. pp. 108–115. IEEE (2012)

Appendix

Security bound for the general construction from Dodis *et al.*

Theorem 2: Security Bound for the general construct from Dodis *et al.* [7] :
A binary linear $[n, k, 2t+1]$ code used in a robust sketch as defined in [7] that does not fulfill the following bound is not provably secure according to the proof provided in [7]:

$$\frac{k}{2} \geq log\left(\sum_{i=1}^{t} \binom{n}{i}\right) + log\left(2\left\lceil \frac{k}{n-k} + 2 \right\rceil\right) - 1 \qquad (10)$$

Proof: Our Theorem is based on Theorem 3 from [7]. In [7] slightly different notions are used than in this paper and the paper from Boyen [2]. In particular the new notion of pre- and post-application robustness is introduced. We will not discuss these definitions and refer the interested reader to [7]. A few variables used in Theorem 3 [7] can be confusing since we use them differently in this paper and therefore we marked variables from Theorem 3 [7] that have a different meaning in this paper with a tilde. Furthermore, v is denoted in [7] with B. In Theorem 3 from [7] it is stated that for any ϵ, δ satisfying

$$l \leq 2m - n - \tilde{k} - 2 \cdot \max\left\{ log(v) + log\left(2\left\lceil \frac{n-\tilde{k}}{\tilde{k}} + 2 \right\rceil\right) + log\left(\frac{1}{\delta}\right), 2log\left(\frac{1}{\epsilon}\right) \right\} \qquad (11)$$

(Gen, Rec) is an $(m, l, t, \tilde{\epsilon})$- fuzzy extractor for M with pre-application robustness $\tilde{\delta}$. The pre-application robustness is basically the chance that an active attacker can perform a helper data manipulation attack while $\tilde{\epsilon}$ is the success probability of a passive attacker. Hence, $\epsilon = \tilde{\delta}$ as we denoted the attack probability of an active attacker with ϵ in this paper. For a linear $[n, k, 2t-1]$ code $\tilde{k} = n-k$ and the maximum possible entropy m is $m = n$. The variable l is the length of the resulting key in bits. We are again interested in an impossibility result that shows that codes not fulfilling the bound cannot be secure (but again fulfilling the bound does not necessarily mean the robust fuzzy extractor is secure). One can bound the attackers success probability ϵ with:

$$\begin{aligned} l &\leq 2m - n - \tilde{k} - 2(log(v) + log\left(2\left\lceil \frac{n-\tilde{k}}{\tilde{k}} + 2 \right\rceil\right) + log\left(\frac{1}{\epsilon}\right) \\ \frac{l}{2} &\leq \frac{k}{2} - log\left(\sum_{i=1}^{t} \binom{n}{i}\right) - log\left(2\left\lceil \frac{k}{n-k} + 2 \right\rceil\right) + log(\epsilon) \qquad (12) \\ log(\epsilon) &\geq log\left(\sum_{i=1}^{t} \binom{n}{i}\right) + log\left(2\left\lceil \frac{k}{n-k} + 2 \right\rceil\right) - \frac{k}{2} - \frac{l}{2} \end{aligned}$$

Since $l \geq 1$ and $\epsilon \leq 1$ we can define the following bound that needs to be fulfilled:

$$\begin{aligned} log(1) \geq log(\epsilon) &\geq log\left(\sum_{i=1}^{t} \binom{n}{i}\right) + log\left(2\left\lceil \frac{k}{n-k} + 2 \right\rceil\right) - \frac{k}{2} - \frac{l}{2} \\ \frac{k}{2} &\geq log\left(\sum_{i=1}^{t} \binom{n}{i}\right) + log\left(2\left\lceil \frac{k}{n-k} + 2 \right\rceil\right) - \frac{1}{2} \qquad (13) \end{aligned}$$

□

Dr. Georg T. Becker is a senior researcher at the Digital Society Institute (DSI) at the ESMT Berlin. Before joining the DSI in December 2016 he was a Post-Doc in the Embedded Security Group at the Horst Görtz Institute for IT-Security at the Ruhr Universität Bochum. His primary research interest is IT-security with special focus on hardware security with topics such as hardware Trojans, Physical Unclonable Functions (PUFs) and side-channel analysis. He received his Ph.D. in Electrical and Computer Engineering at the University of Massachusetts Amherst in 2014 and holds a M.Sc. degree in IT-Security (2009) and a B.Sc. degree in Applied Computer Science (2007) from the Ruhr Universität Bochum.