Breaking through Fixed PUF Block Limitations with Differential Sequence Coding and Convolutional Codes

Matthias Hiller, Michael Weiner, Leandro Rodrigues Lima,
Maximilian Birkner, and Georg Sigl
Institute for Security in Information Technology
Technische Universität München, Munich, Germany
{matthias.hiller, m.weiner, sigl}@tum.de
{gu27wik, maxi.birkner}@mytum.de

ABSTRACT

Secret key generation with Physical Unclonable Functions (PUFs) is an alternative to conventional secure key storage with non-volatile memory.

In a PUF, secret bits are generated by evaluating the internal state of a physical source. Typically, error correction is applied in two stages to remove the instability in the measurement that is caused by environmental influences.

We present a new syndrome coding scheme, called Differential Sequence Coding (DSC), for the first error correction stage. DSC applies a fixed reliability criterion and searches the PUF output sequence sequentially until a number of suitable PUF outputs is found. This permits to guarantee the reliability of the indexed PUF outputs. Our analysis demonstrates that DSC is information theoretically secure and highly efficient.

To the best of our knowledge, we are the first to propose a convolutional code with Viterbi decoder as second stage error correction for PUFs. We adapt an existing bounding technique for the output bit error probability to our scenario to make reliability statements without the need of laborious simulations.

Aiming for a low implementation overhead in hardware, a serialized low complexity FPGA implementation of DSC and the Viterbi decoder is used in this work.

For a reference SRAM PUF scenario, PUF size is reduced by 20% and the helper data size decreases by over 40% compared to the best referenced FPGA implementations in each class with a minor increase in the number of slices.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection
; E.4 [**Coding and Information Theory**]: Error Control Codes

Keywords

Physical Unclonable Functions (PUFs); Syndrome Coding; Error Correction; Differential Sequence Coding (DSC); Fuzzy Extractor; Convolutional Code; Viterbi Decoder.

1. INTRODUCTION

Physical Unclonable Functions (PUFs) generate unique secrets from manufacturing variations in integrated circuits. They enable the use of cryptographic algorithms and secure authentication protocols for devices manufactured in standard technologies without access to secure non-volatile memory. In this work, we will focus on secure key generation with PUFs. Meanwhile, PUFs were presented for several application scenarios such as RFID [1] and vehicular security [2], secure storage [3] and also intellectual property protection [4, 5, 6]. PUFs can even be used in high security devices, such as smart cards [7].

Several PUF types evaluate different physical measures, such as time, voltage, resistance, or capacitance to derive a unique internal secret. It is clearly visible in overviews, e.g. [8, 9, 10, 11], that a large amount of previous work and ongoing research is dedicated to the problem to generate reliable and unpredictable binary information from physical structures.

A *PUF output* is a physical structure that reveals the *PUF response* as result of a measurement of specific internal properties of this structure. In the following, we will present generic work that is independent of a specific PUF implementation. Therefore, we abstract from a specific type or implementation and interpret the PUF output as a random variable and the PUF response as its outcome.

An ideal PUF behaves like a memory programmed with a random number. However, to date no PUF structure is known that provides perfect reliability and randomness at the same time.

Thus, *syndrome coding schemes*, e.g. [12, 13] are necessary to derive reliable cryptographic keys from noisy PUF responses. All syndrome coding schemes have in common that a public *syndrome* or *helper data* is generated during an enrollment process. In channel coding, the syndrome is defined as product of a received codeword with the parity check matrix [14]. It is used to find the errors in a received codeword but does not contain information about the codeword itself.

In our application, the syndrome serves a similar purpose but is computed differently. It contains information on the properties of the corresponding specific PUF instantiation to detect and eliminate variations among different PUF responses of this specific PUF. This permits to reproduce exactly the same secret from a similar subsequent PUF response and the helper data. However, since it is public the helper data must not reveal any substantial information about the secret.

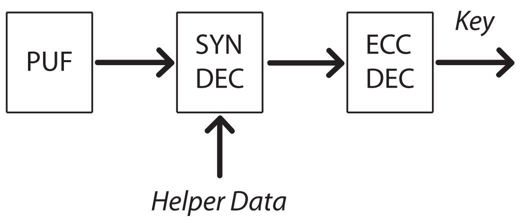

Figure 1: Concatenation of a syndrome and ECC decoder for secret key reproduction with PUFs

Typically, the key reproduction is carried out in two stages. The syndrome coding is concatenated with error-correcting codes (ECC), as shown in Figure 1. This enables to derive keys with arbitrarily small bit error probabilities from noisy PUF responses. *Fuzzy extractors* [12] are built from the same core building blocks with the limitation that the secret key is determined by the device rather than the user.

Several publications on different scenarios and code classes, e.g. [4, 15, 16, 17, 13, 18, 19, 20, 21], address a wide range of interesting aspects. Later on, we will use the results presented in [15, 16, 17, 18] as reference because all four are based on the same reproducible scenario introduced in [4].

To the best of our knowledge, all previous syndrome coding schemes were concatenated with block codes [14]. Referenced implementations map a fixed number of secret bits to a fixed blocks of PUF outputs. We treat all PUF outputs as one long sequence and select a fixed number of PUF outputs with specific properties within a sequence of variable length. Distance values between the selected PUF outputs are stored as syndrome.

Our goal in designing DSC was to provide a low complexity algorithm ensuring that no unreliable PUF outputs are used for key reproduction.

Looking at the second error correction stage, the selection of the code class is the first crucial step in applying error-correcting codes in any scenario. It has a great impact on the set of available code parameters and the resulting bit error probability. Ultimately, also the implementation complexity varies greatly for different code classes.

We suggest to use a convolutional code for the first time in the context of PUFs. Convolutional codes are known for their good error correction performance and low hardware overhead. They play an important role in channel coding, especially in the early days of coding theory [22]. The existence of the Viterbi algorithm [23], a highly efficient decoder whose complexity only scales linearly with code length, makes convolutional codes attractive for implementation.

By combining both sequence based syndrome and channel coding, we present a holistic fuzzy extractor with new and efficient core building blocks.

Our contributions

- Introduction of our new sequence-based syndrome coding scheme DSC
- Performance and security analysis of DSC
- First usage of convolutional codes in the PUF context
- Fuzzy extractor FPGA implementation and synthesis for two popular entry level FPGAs
- PUF size reduction by 20% and helper data saving of 40% at a slightly increased decoder complexity for an SRAM PUF scenario

Outline

The introduction is followed by a brief survey on related work in Section 2 and the definitions used in this work in Section 3. Section 4 highlights the shortcomings of previous work and motivates our new approach DSC, which is introduced in Section 5. The performance of DSC is analyzed in Section 6 and the security is addressed in Section 7. Section 8 gives an introduction to convolutional codes. We discuss the performance of our fuzzy extractor in Section 9 and present an FPGA implementation in Section 10.

2. RELATED WORK

The *code-offset fuzzy extractor* by Dodis *et al.* [12] is based on the fuzzy commitment by Juels and Wattenberg [24]. It masks a codeword by an XOR operation with the entire PUF response. Early work by Suh [25] already used a Bose-Chaudhuri-Hochquenghen (BCH) code [14]. Several hardware implementations covered different aspects and goals. Bösch *et al.* [15, 26] evaluated Reed–Muller, BCH and Golay codes [14] for a hard decision scenario[1]. In [16, 17], Maes *et al.* presented a soft decision approach with Reed–Muller codes and decoding as generalized multiple concatenated codes by Schnabl and Bossert [27]. Storing reliability information in the helper data increased the efficiency compared to previous work.

Further, Maes *et al.* [19] used BCH codes with a highly optimized BCH decoder [28] for a stand-alone hardware implementation that also contained a ring oscillator PUF. Van der Leest *et al.* analyzed small Reed–Muller codes with brute force decoding and a Golay code decoder for soft decision decoding in [20]. Here, the soft information is derived from a single read-out of the SRAM cells.

Instead of connecting the PUF response and a codeword with XOR operations, *Index-Based Syndrome Coding* (IBS) by Yu and Devadas [13] searches a PUF output block for a PUF output whose response equals most likely the value of a given codeword bit and stores a pointer to this PUF output. Yu *et al.* [21], evaluated PUF ASICs using IBS and BCH codes with an Euclidean solver and Chien search [14]. The ASICs showed a large reliability over varying environmental conditions.

Complementary IBS (C-IBS) by Hiller *et al.* [18] increases the reliability of IBS by adding an additional encoding step and applying IBS iteratively multiple times to the same PUF output block.

[1]For hard decision decoding, all codeword bits contribute equally to decoding. Soft decision decoding takes additional reliability information into account to improve the error correction performance.

Yu *et al.* compared different approaches and implementations in [29]. Other recent error correction and authentication schemes can be found in [30] and [31].

3. DEFINITIONS AND NOTATION

In the following, random variables are denoted in capital italic letters, e.g. R, with outcomes in small italic letters, e.g. r.

Further, a superscript over a random variable, e.g. R^n, denotes n random variables with the same output alphabet. Note that we do not apply any further restrictions on the Rs so that the random variables in R^n can have different probability distributions. The probability of event A is represented by $\Pr[A]$.

To distinguish between channel and syndrome coding, syndrome coding is addressed with parameters in Latin letters, whereas the corresponding Greek letters are used for channel coding. A (ν, κ, δ) code has codewords of length ν. κ information bits are encoded to one codeword and the codewords have a minimum Hamming distance of δ to each other. Thus, the code can correct at least $\left\lfloor \frac{\delta-1}{2} \right\rfloor$ errors.

4. MOTIVATION

Block codes can correct a specific number of bit errors in every codeword. Large codewords are favorable for error correction because the distribution of errors in every codeword approaches the distribution of errors closer and closer[2].

In our application, the error probability of every code bit depends on the error probability of the PUF output(s) it is linked to. In contrast to independent and identically distributed (i.i.d.) errors in most communication scenarios, our codewords have different error probabilities depending on the specific PUF instantiation, even if the PUF outputs are i.i.d..

The existing syndrome coding schemes generate helper data with the goal to minimize the error probability of a reproduced key sequence for a fixed given number of PUF outputs. The error probability depends on the distribution of the reliability of the PUF outputs within the given block.

Example:

The simulation of a code-offset syndrome coding scheme with a $(3, 1, 3)$ repetition code with soft-decision maximum likelihood decoding and an SRAM PUF with an average bit error probability of 15% according to [16] leads to an average output bit error probability of 3.1%.

However, the cumulative distribution function shows that one third of the PUF outputs has bit error probabilities $<$ 2.9%.

Therefore, selecting the most reliable third of the PUF outputs without any additional error correction would reduce the output bit error probability with the same number of key bits per PUF output.

Based on this observation, we will introduce a new syndrome coding scheme with the goal to select the most reliable PUF outputs of a long sequence and find an efficient syndrome representation.

[2]Ultimately this leads to the Channel Coding Theorem [32], that says that the error probability can be brought down to zero by spending a specific minimum amount of redundancy, quantified by the *channel capacity*, and a code length going to infinity.

5. DIFFERENTIAL SEQUENCE CODING

We define a reliability criterion and guarantee that only PUF outputs will be used that fulfill this criterion. We fix the reliability and vary the block size in contrast to previous work, where the block size was fixed and the reliability varied.

DSC searches the sequence of PUF outputs for PUF outputs with a bit error probability smaller than an a priori defined probability p_{max}. The DSC algorithm is inspired by design principles in lightweight cryptography (e.g. [33]). We split the algorithm into several iterations with low data dependency and complexity. Other than e.g. C-IBS [18], our syndrome coding algorithm is separated into independent iterations for every secret bit that do not share intermediate or even output data. Ending one iteration only provides the starting point for the next iteration and triggers the execution.

Our approach DSC shows parallels to labeling unreliable PUF outputs with dark bits by Armknecht *et al.* [34]. Instead of removing unreliable bits and applying syndrome coding afterwards, both steps are combined in DSC. The simplified version of DSC without inversion bit in the next subsection still shows the properties of a secure syndrome coding scheme.

We will start with a toy example to demonstrate the intuition of the algorithm and then give a precise specification for implementation. The full encoding algorithm is presented in Algorithm 1.

5.1 Example

In Figure 2 zeros are marked with white boxes, and ones are marked with black boxes. p_{max} is the highest tolerated error probability. PUF output R, with probability $\Pr[r = 0] \geq 1 - p_{max}$ is also denoted with a white box. For black boxes, $\Pr[r = 1] \geq 1 - p_{max}$ and gray boxes stand for unreliable PUF outputs with $p_{max} < \Pr[r = 1] < 1 - p_{max}$. In this example, code sequence $c^4 = (0, 1, 0, 1)$ is embedded into the PUF response.

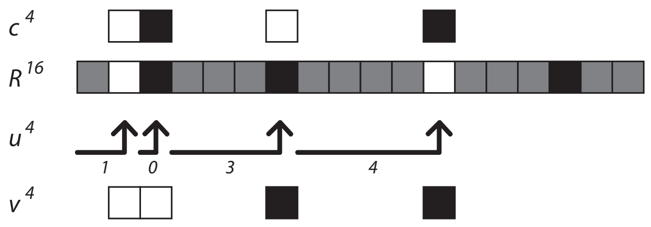

Figure 2: Example for DSC encoding

The goal of a simplified first DSC iteration is to find the first PUF output with the same color as the first code sequence bit and store a relative distance pointer to the starting in the helper data. The "code box" and the next reliable "PUF box" in the sequence have the same color with a probability of 50%, so only 50% of the reliable PUF outputs are indexed.

All reliable PUF outputs can be indexed by adding an additional inversion bit that indicates if the boxes have the same color. Therefore, the relative distance pointer u_1 references the distance between the starting point and the first reliable PUF output. Inversion bit v_1 represents if the "code box" and the indexed "PUF box" have the same color, or not.

R_2 is the first PUF output within the specification, so $u_1 = 1$. $\Pr[r_2 = c_1] \geq \Pr[r_2 = c_1 \oplus 1]$, so no inversion is necessary and $v_1 = 0$. The helper data $w^4 = (u^4, v^4)$ is computed after four iterations such that $u^4 = (1, 0, 3, 4)$ and $v^4 = (0, 0, 1, 1)$.

5.2 Encoding Algorithm

DSC has two major parameters that specify the sizes of input and output data of DSC: k is the length of the embedded sequence and n the maximum length of the indexed PUF sequence. The cryptographic key is encoded to a code sequence c^k in the encoder of the error-correcting code. Then, every bit of the code sequence is mapped to one PUF output in the DSC encoder and the differential mapping is stored as syndrome.

The encoding algorithm presented in Algorithm 1 searches n PUF outputs R^n with PUF responses r^n. Therefore, PUF outputs R are selected such that $\Pr[r = 1] \geq 1 - p_{max} \vee \Pr[r = 0] \geq 1 - p_{max}$ for a fixed maximum error probability $p_{max} > 0$. The helper data sequence w^k contains the syndrome. Every element $w_i = (u_i, v_i)$, $i = 1, ..., k$, contains the distance $u_i \in \{0, ..., 2^l - 1\}$, $l \in \mathbb{N}$, and an inversion bit $v_i \in \{0, 1\}$.

Algorithm 1: DSC Encoding

```
$o := 0$ *(The offset counter $o$ tracks absolute the position within $R^n$)*
**for** $i := 1 \to k$ **do**
  *Search for one PUF output for each code sequence bit.*
  **for** $j := 1 \to 2^l$ **do**
    **if** $o + j > n$ **then**
      Return $error\_1$ *(Not enough PUF outputs within the specification)*
    **else if**
    $\Pr[r_{j+o} = 0] \geq 1 - p_{max} \vee \Pr[r_{j+o} = 1] \geq 1 - p_{max}$
    **then**
      $u_i := j - 1$
      **if** $\Pr[r_{j+o} = c_i] \geq \Pr[r_{j+o} = c_i \oplus 1]$ **then**
        $v_i := 0$ *(No inversion)*
      **else**
        $v_i := 1$ *(Inversion)*
      **end if**
      $o := o + j$
      Break
    **else if** $j = 2^l$ **then**
      Return $error\_2$ *(Counter overflow)*
    **end if**
  **end for**
**end for**
```

5.3 Discussion

Section 4 demonstrated variations in the reference approaches that operate on small fixed block sizes. We treat the entire PUF output vector as one single sequence to minimize the statistical variation. Due to the law of large numbers and the large sample size inside the single sequence, we are able to predict the statistics of the PUF outputs inside this sequence with a small error.

As a consequence, this permits to predict the existence of PUF outputs with specific properties with a high confidence. However, the positions of the PUF outputs with the specific properties are unknown.

The goal of the presented syndrome coding scheme is to take advantage of the precise knowledge over the statistics while minimizing the required resources as well as the implementation complexity.

6. PERFORMANCE

Two measures characterize the efficiency of DSC:

- The yield corresponds to the probability that valid helper data can be generated for a given parameter set and specific PUF type.
- The bit error probability defines the reliability during operation.

This section shows the trade-off between taking a small and well controlled risk that a device will fail during production, and reducing the error probability of the device out in the field.

During manufacturing and testing, devices are easy to access and the consequences of a particular device that does not fulfill a required reliability specification are limited. It can be discarded easily within the manufacturing process. When such a device is deployed in the field, the operation outside the specification threatens the integrity of the system in which the device is used. To make things worse, it can be very hard to detect the violation of the specification in the field without the sophisticated test equipment that is available during manufacturing. Even when it is detected, it can still be difficult to replace the device in a timely manner. Therefore, designers are likely willing to trade a reasonable loss in yield during manufacturing for an increased and well controlled reliability in the field. As addressed later in this section, DSC gives the designer the chance to balance between assigned resources and yield.

IBS and C-IBS maximize the reliability of the key for a given PUF instantiation. For the referenced code-offset implementations, all PUF outputs are used regardless of their reliability. The average error probabilities are well controlled for all approaches, but the error probability of a specific device is not addressed during manufacturing. This can lead to the problem that an unreliable device is deployed in the field even if the batch of devices fulfills the required mean error probability.

DSC evades this issue by ensuring a minimum reliability for every device, for which Algorithm 1 terminated without errors.

Let $pdf(\cdot)$ define the probability density function and $cdf(\cdot)$ the cumulative distribution function over all PUF outputs. Further, let $\Pr[\cdot]$ denote a probability and $E(\cdot)$ the expectation. We define p as the probability that the error probability of a PUF output R is within the specified range.

$$p = \Pr[E(R) \in \{[0, p_{max}] \cup [1 - p_{max}, 1]\}] \quad (1)$$
$$= cdf(p_{max}) + (1 - cdf(1 - p_{max})) \quad (2)$$

A precise performance estimation for each PUF output, as required for the soft decision decoders [16] and [18] or the

IBS encoding process in [13] and [18] , adds an additional effort. For DSC, the effort is reduced because DSC only makes a binary decision, if the reliability of a given PUF output is better than a given threshold, or not. In practice, reliability information can be obtained through multiple measurements [16] or statistical models based on soft PUF output information [35]. Special SRAM circuits also permit to assess the reliability of SRAM cells with a low number of measurements [36, 37]. The counters in ring oscillator PUFs can be used to generate soft information as side effect [38].

6.1 Yield

There are two events in which Algorithm 1 fails to generate helper data: The first event *error_1* occurs when a PUF does not contain enough PUF outputs with the specified maximal error probability p_{max} or less. The second event *error_2* occurs when the distance between two indexed PUF outputs exceeds the maximum value that can be represented in the helper data.

The probability e_1 that *error_1* occurs is the sum over all probabilities for outcomes in which R^n contains less than k PUF outputs with expectation $E(R_i) \in \{[0, p_{max}] \cup [1 - p_{max}, 1]\}$, $i \in \{1, ..., n\}$.

$$e_1 = \sum_{i=0}^{k-1} \binom{n}{i} p^i (1-p)^{n-i} \tag{3}$$

For the second event *error_2* and maximum distance value $q = 2^l$, no suitable PUF output is found for a single code sequence bit with probability $(1-p)^q$. Thus, no code sequence bit will fail with $(1-(1-p)^q)^k$. As complementary event, the probability of at least one overflow in a code sequence is

$$e_2 = 1 - (1 - (1-p)^q)^k \tag{4}$$

As a consequence, the yield y is computed by the probability that neither *error_1* nor *error_2* occur. *error_1* and *error_2* are not disjoint, so

$$y > 1 - (e_1 + e_2) \tag{5}$$

Note that e_1 defines the correlation between worst case reliability and size of the PUF, whereas e_2 is only affected by the size constraint of the helper data.

The events *error_1* and *error_2* define hard break conditions and affect the yield directly. The sum over all distance values $\sum_{i=1}^{k} u_i$ reveals another quality measure to evaluate the quality of the PUF. Over a maturing manufacturing process, the average distance u_i should decrease. Therefore, n could be reduced for later designs in the matured manufacturing process while keeping y at the initial level.

6.2 DSC Bit Error Probability

The maximum output bit error probability of DSC is given by p_{max}. The mean error probability of DSC p_{err} is calculated by the integral over the error probabilities of all PUF outputs within the specification, weighted by their probability of occurrence, so

$$p_{err} = \frac{1}{p} \left(\int_{0}^{p_{max}} pdf(x) \cdot x \; dx + \int_{1-p_{max}}^{1} pdf(x) \cdot (1-x) \; dx \right) \tag{6}$$

For a fair and sound comparison with previous work, the syndrome coding schemes are first compared without additional error-correcting codes and the same SRAM PUF probability distribution presented in [16]. For optimal performance of all schemes, we assume that reliability information is available during the helper data generation. Later on, all schemes including DSC will only require one binary response for each PUF output. Note that DSC will only use soft information only during helper data generation whereas related work [17, 18] always relies on soft information during helper data generation and also key reproduction.

A comparison of DSC with simulation results of previous approaches demonstrates its efficiency. We chose to concatenate the code-offset syndrome coding with a repetition code (CO+REP) so that all approaches reproduce one key bit per block. We chose [16] as reference PUF output distribution to generate comparable results. Recent work in this direction [39, 35] presented more precise approaches to derive the bit error probability distribution of a PUF.

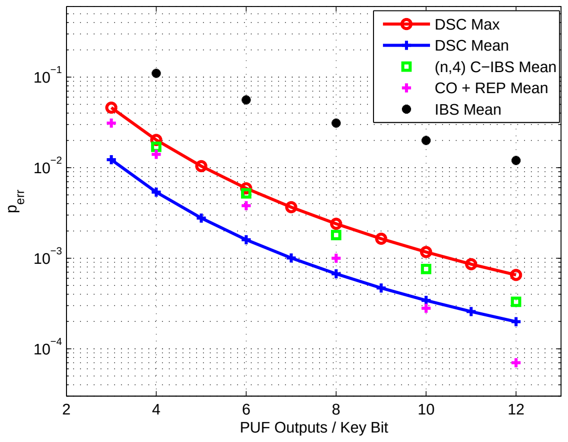

Figure 3: Max and mean bit error probabilities of syndrome coding schemes without second stage ECCs for SRAM PUFs according to [16] with average bit error probability 15%. For DSC and key size $s = 128$, n is chosen such that $e_1 = 5 \cdot 10^{-4}$. Reference values were taken from [40].

Figure 3 shows the bit error probabilities of different syndrome coding schemes over the number of required PUF outputs per key bit. The maximum error probability p_{max} of DSC is comparable to the mean error of previous approaches for small PUF output to key bit ratios. We obtained the p_{max} values for DSC by minimizing p_{max} for a fixed n under the constraint of a predefined yield given by Equation 3. All data points of related work show mean error probabilities.

For the mean bit error probability p_{err}, DSC shows less errors than C-IBS and the code-offset syndrome coding with a repetition code for a ratio of 4 PUF outputs per key bit. The increased reliability of DSC is caused by the effect that the reliability of the blocks varies for the block based approaches. The curves of the DSC mean error and the code-offset cross for $\frac{n}{k}$ between 9 and 10. This demonstrates that selecting around 10% of the bits very carefully with DSC is as efficient as combining blocks of 9 with the code-offset method.

7. SECURITY ANALYSIS

The security analysis contains a theoretical and a practical part. The information theoretic analysis quantifies the amount of key information that leaks through the helper data and the helper data attack shows threats to hardware implementations of the DSC decoding algorithm.

7.1 Information Theoretic Analysis

For code sequence C^k, PUF outputs R^n and helper data W^k, the mutual information[3] between the code sequence and the helper data $I(C^k;W^k)$ determines the amount of key information that leaks through the helper data. Let $\overline{R}^k$ describe the vector of selected PUF outputs in R^n. According to the definition of the mutual information,

$$I(C^k;W^k) = H(C^k) - H(C^k|W^k) \tag{7}$$

The helper data W_i is computed as a function f of the previous helper data W^{i-1}, the current code sequence bit C_i and the selected PUF output $\overline{R}_i$.

$$W_i = f(W^{i-1}, (C_i \oplus \overline{R}_i)) \tag{8}$$

The distance values U^k are selected independently from the key, so they cannot leak any key information. Therefore, the leakage of the helper data W^k only depends on

$$V^k = C^k \oplus \overline{R}^k \tag{9}$$

Using the helper data computation in the conditioned entropy in Equation 7 gives

$$H(C^k|W^k) = H(C^k|C^k \oplus \overline{R}^k) \tag{10}$$

$$= H(C^k, C^k \oplus \overline{R}^k) - H(C^k \oplus \overline{R}^k) \tag{11}$$

$H(C^k \oplus \overline{R}^k|C^k) = H(\overline{R}^k)$, which removes the XOR in the joint entropy, such that

$$= H(C^k, \overline{R}^k) - H(C^k \oplus \overline{R}^k) \tag{12}$$

C^k and $\overline{R}^k$ are independent according to the problem definition, so

$$= H(C^k) + H(\overline{R}^k) - H(C^k \oplus \overline{R}^k) \tag{13}$$

Using this result in Equation 7 gives

$$I(C^k;W^k) = H(C^k \oplus \overline{R}^k) - H(\overline{R}^k) \tag{14}$$

[3]In the following, we will use the definitions and notation for entropy $H(\cdot)$, joint entropy $H(\cdot,\cdot)$, conditional entropy $H(\cdot|\cdot)$ and mutual information $I(\cdot;\cdot)$ according to [32].

An upper bound can be given by $H(C^k \oplus \overline{R}^k) \leq k$, so

$$I(C^k;W^k) \leq k - H(\overline{R}^k) \tag{15}$$

In general, correlated PUF outputs can lead to syndromes which leak key information if too much key information is stored. The main insight from Equation 14 is that the leakage can be reduced nearly down to zero with diligent code design such that $H(C^k \oplus \overline{R}^k) \leq H(\overline{R}^k) + \epsilon$ for a small $\epsilon > 0$. Therefore, the codes have to be designed in such a way that the bias or correlations in the PUF are also represented in the code structure.

According to the bound in Equation 15, DSC is information theoretically secure for any code if the PUF has a high entropy such that $H(\overline{R}^k) > k - \epsilon$. According to [41], this is given for the SRAM PUF.

7.2 Helper Data Manipulation Attack

In our system model, we assume that an attacker has no access to the PUF outputs or the key. However, the helper data can be stored in an unprotected external memory, so an attacker can arbitrarily read or modify this data. Furthermore, we assume that the attacker can verify if a cryptographic operation cryptop(*data_input*, *key*) that uses the generated key from the PUF response shows valid behavior or not, for example by observing if a firmware decryption is successful and the system boots properly.

In previous pointer based syndrome coding schemes such as IBS or C-IBS, independent blocks of PUF outputs were used for each codeword bit. DSC does not split the PUF response into blocks but uses one long sequence out of which *all* code sequence bits are referenced.

From a security perspective, having one long sequence, i.e. one address space, for all PUF outputs is problematic: it allows an attacker to compare different PUF response bits by modifying the helper data. As a result, he can learn whether PUF response bits corresponding to c_i and c_{i+1} are equal or the inverse of each other.

In a simplified scenario, we assume that no second stage error correction is used after the syndrome coding. The error correcting-code is not required for a successful attack and the simplified scenario makes the problem accessible more easily. As a consequence, the term "codeword" is replaced by "key" for the attack description.

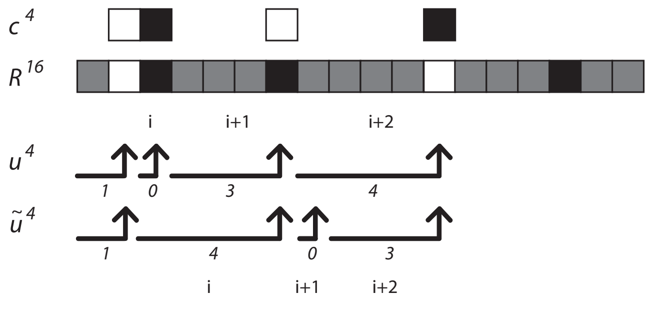

Figure 4: Example for Helper Data Manipulation

Figure 4 shows the attack strategy. To attack our scheme, the attacker manipulates the helper data u^4. In the changed helper data $\tilde{u}^4$, he shifts pointer i to point to position $i+1$, then modifies pointer $i+1$ to point to an unused bit between $i+1$ and $i+2$ and finally adjusts the distance of pointer $i+2$ such that position $i+2$ and all subsequent pointers are addressed correctly again. The fact that the unused bit is not part of the key implies that its stability is below the required threshold.

Therefore, the attacker can assume that if several DSC key extractions are performed, this bit will be equal to key bit c_{i+1} in some of the extractions. Of course, the attacker cannot observe in which of the extractions this is the case; however, for those cases it becomes important whether $c_i = c_{i+1}$ holds or not. If it holds, the extracted key is equal to the original one.

The attacker can evaluate whether there exists a significant number of unchanged keys by observing the output of a cryptographic operation or by verifying whether a cryptographic operation such as firmware decryption is successful. Eventually, the attacker can repeat this procedure for every pair of subsequent bits such that in the ideal case, only one key bit remains unknown independently of the actual key length.

For a successful attack, it is required to modify single targeted key bits while keeping the rest at its original value. Therefore, the attack can be prevented if the attacker cannot address single key bits anymore. For this reason, we chose to integrate a hash function into the proposed scheme. The output of the error correction code is XORed with the hash value of the helper data; with this addition, any change in the helper data will on average lead to a change of 50% of the key bits. Note that in our construction, only the public helper data is fed into the hash function such that it is not threatened by side-channel attacks such as [42].

We assume that the system in which the fuzzy extractor is integrated is able to handle invalid keys. For example, the system can go into an error state if the boot code is not decoded correctly. For systems in which our assumption does not hold, robust fuzzy extractors [43, 44] are able to detect helper data manipulations. However, the proposed robust fuzzy extractors require additional decoding effort, PUF bits and helper data.

8. CONVOLUTIONAL CODES

After addressing syndrome coding in detail in the previous sections, this section provides an introduction to convolutional codes and presents the error-bounding technique that will be used for evaluation.

For all classes of channel codes, an information sequence is encoded to a longer code sequence with code specific structure. The code sequence is exposed to an environment that potentially changes parts of the sequence. On the decoder side, errors in a given sequence are corrected by detecting distortions that contradict with the code structure and solving these contradictions. To avoid confusions between channel and syndrome coding, we will denote all channel code parameters with the corresponding Greek letters.

Block codes divide the information sequence into blocks that are encoded independently into code blocks. Increasing the block length improves the error correction performance, but also increases the decoding complexity.

As one major advantage of convolutional codes, there exist very efficient decoding algorithms [14]. Convolutional codes offer powerful error correction for a low hardware overhead. In this work, we will focus on the Viterbi algorithm [23].

Similar to a window sliding along the information sequence, every code sequence bit depends on a constant number of consecutive information bits. This operation is defined mathematically as convolution.

8.1 Convolutional Encoder

The encoder of an $(2, 1, [\mu])$ code encodes one input sequence to 2 output sequences c_1, c_2 with

$$c_1 = c_{1_1} c_{1_2} c_{1_3} \ldots$$
$$c_2 = c_{2_1} c_{2_2} c_{2_3} \ldots$$

Output sequences c_1 and c_2 are concatenated to the code sequence c according to

$$c = c_{1_1} c_{2_1} c_{1_2} c_{2_2} \ldots$$

The convolution operation is carried out in hardware by shifting the information sequence through a shift register of length μ, as shown in Figure 5.

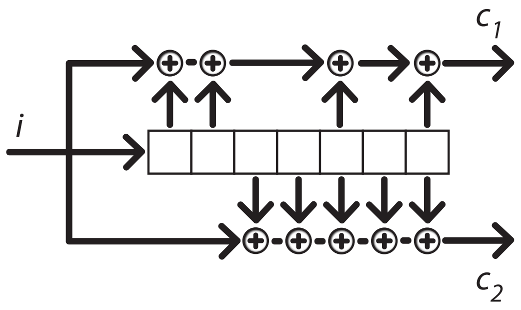

Figure 5: $(2, 1, [7])$ **convolutional encoder**

Functions of the input bit i_j and the internal state of the decoder $(i_{j-1}, \ldots, i_{j-\mu})$ compute two output bits (c_{1_j}, c_{2_j}) for each input bit. Later in this paper, we will use a $(2, 1, [7])$ code with

$$c_{1_j} = i_j \oplus i_{j-1} \oplus i_{j-2} \oplus i_{j-5} \oplus i_{j-7}$$
$$c_{2_j} = i_j \oplus i_{j-3} \oplus i_{j-4} \oplus i_{j-5} \oplus i_{j-6} \oplus i_{j-7}$$

As main difference to block codes, every information bit is woven into the code stream with a certain impact length μ instead of encoding independent blocks where every code bit might be affected by every information bit within the same block. After the last information bit, μ zeros are shifted into the encoder so that the last information bit also affects the specified number of code sequence bits.

A larger memory increases the number of code sequence bits that are affected by every information bit, which increases the error correction capability of the code, as well as its implementation complexity. A comprehensive introduction and analysis of convolutional codes and the Viterbi algorithm that will be used in our hardware implementation can be found in [14].

8.2 Bounding the Bit Error Probability of Convolutional Codes

The bit error probability is a key parameter to evaluate the performance of a code in a given scenario. It is computationally infeasible to calculate the bit error probability for most cases. So, the straight forward approach is to run Monte Carlo simulations until a significant number of errors is detected and calculate the average bit error probability. For low bit error probabilities, simulations can become very time and resource consuming.

Bounding techniques simplify a given problem such that it becomes feasible to compute a bound that gives a best or a worst case statement. The following upper bound for the bit error probability of convolutional codes is based on the bound discussed by Bossert in [14].

The convolutional code is a linear code. According to the definition of linear codes, the sum of two code sequences is a code sequence and the all zeros sequence also is a code sequence.

Using this linearity property, we can analyze the probability of a decoding error of the decoding of any random code sequences to any other code sequence by referring the sum of both sequences to the all zeros sequence. Therefore, we can characterize the behavior of the entire code from the all zeros sequence.

For the all zeros code sequence, a decoding error occurs in a sequence with Hamming weight w if e bit errors occur with

$$e > \lfloor (w-1)/2 \rfloor \tag{16}$$

The probability p_w of a decoding error in a sequence with Hamming weight w for a channel with bit error probability p is bounded by

$$p_w < \left(2\sqrt{p(1-p)}\right)^w \tag{17}$$

To analyze the mean error over the PUF output distribution, the expectation over the distribution is calculated.

$$E(p_w) < E\left(\left(2\sqrt{p(1-p)}\right)^w\right) \tag{18}$$

For i.i.d. PUF outputs every factor can be treated independently.

$$E(p_w) < \left(E\left(2\sqrt{p(1-p)}\right)\right)^w \tag{19}$$

For the further bounding we will apply Jensen's inequality [32] twice on concave functions. The concavity of the square root function permits

$$E(p_w) < \left(2\sqrt{E\left(p(1-p)\right)}\right)^w \tag{20}$$

Again, $p(1-p)$ is concave, so

$$E(p_w) < \left(2\sqrt{E(p)(1-E(p))}\right)^w \tag{21}$$

Using $E(p) = p_{err}$ computed in Equation 6, $E(p_w)$ can be bounded by

$$E(p_w) < \left(2\sqrt{p_{err}(1-p_{err})}\right)^w \tag{22}$$

The information weight $I(w)$ of a code gives the number of sequences with Hamming weight w. The information weight spectrum of the code used in this paper can be found in [45] and the bit error probability is bounded by

$$p_{output\ err} < \sum_w I(w) \cdot E(p_w) \tag{23}$$

As a result, Equation 23 bounds the bit error probability of a convolutional code and thus permits to evade laborious simulations.

9. FUZZY EXTRACTOR BIT ERROR PROBABILITY

This section extends the isolated DSC results presented in Section 6 to demonstrate the performance of the entire fuzzy extractor.

The full fuzzy extractor is evaluated for the same benchmark scenario from [4]. In this scenario, the system is designed to reproduce a 128 bit key with a key error probability of 10^{-6}, which corresponds to a bit error probability of $7.81 \cdot 10^{-9}$. An SRAM PUF with 15% bit error probability is used at the input. Maes *et al.* extended this scenario to an input distribution [16], as it was already applied to the stand alone syndrome coding schemes in a previous section.

For (n,k) DSC concatenated with a $(2,1,[\mu])$ convolutional code and a key size of s, the number of required PUF bits per key bit is given by

$$\frac{n}{s} = \frac{n}{k} \cdot \frac{2(s+\mu)}{s} \tag{24}$$

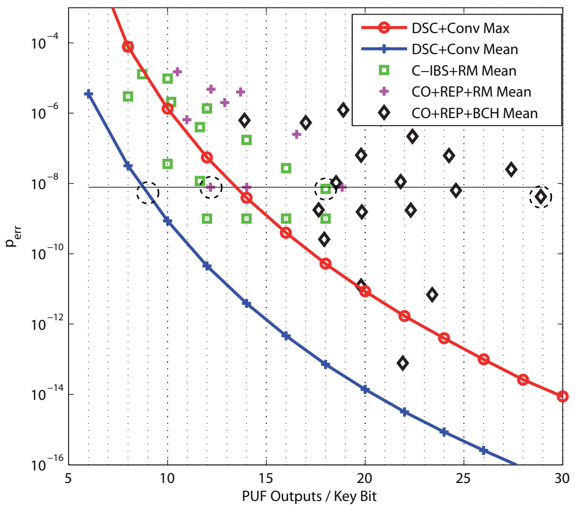

Figure 6: Bounded mean and max key bit error probabilities of DSC concatenated with a $(2,1,[7])$ convolutional code for an SRAM PUF according to [16] with average bit error probability 15%. Again, $e_1 = 5 \cdot 10^{-4}$. Reference values were taken from [26, 16, 40].

Figure 6 shows the relation between the number of PUF outputs per key bit and either bounded or simulated bit error probabilities for several fuzzy extractors. Previous work [26, 16, 40] is represented with points and the bold lines show our new DSC fuzzy extractor. All solutions below the black horizontal line fulfill the mean key error probability of 10^{-6}, but vary in the number of required PUF and helper data bits, as well as the implementation complexity. We bounded the bit error probabilities of the convolutional codes with the method presented in Section 8.2 for the DSC output error probabilities presented in Figure 3 as input.

The good performance of DSC in Figure 3 translates directly to a low number of PUF outputs per key bit in Figure 6.

Again, the red line shows that the maximum error of DSC is comparable to the lower mean bit error probabilities of previous work. The blue line demonstrates that the mean error of the DSC fuzzy extractor is lower than the referenced approaches for a $(2, 1, [7])$ convolutional code of moderate size. The reference approaches [16, 18] on the left side of Figure 6 use soft-decision decoders that require reliability information in the helper data. The convolutional code reaches a similar performance with less complex hard decision decoding.

Our DSC configuration in the dotted circle with $\frac{n}{s} = 9$ has a moderate syndrome coding output bit error probability compared to reference C-IBS and code-offset values. However, the fuzzy extractor output bit error probability is still smaller than $7.81 \cdot 10^{-9}$, which is required for the key error probability of 10^{-6}. This shows that the $(2, 1, [7])$ code is an efficient choice for the given problem.

Hardware implementations of the setups marked by dotted circles will be compared in the next section.

10. FPGA IMPLEMENTATION

The previous sections have demonstrated that DSC is efficient in terms of required PUF outputs and also is provably secure. The implementation complexity is another important benchmark to evaluate the practical value of DSC.

We present an overview over our area optimized low throughput fuzzy extractor FPGA implementation for DSC parameters $n = 9(s+7)$, $p_{max} = 0.0143$, a $(2, 1, [7])$ convolutional code and $s = 128$ bit key size in this section.

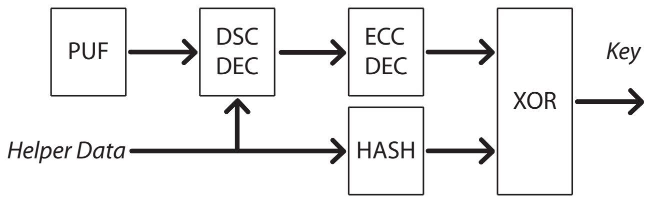

Figure 7: Hardware implementation of DSC fuzzy extractor key reproduction

The implementation contains four modules: The DSC decoder maps the helper data and PUF response to a code sequence. The error-correcting code (ECC) decoder, in this case a Viterbi decoder, computes the embedded information sequence. In order to prevent the helper data manipulation attack in Section 7, the hash function creates a hash value of the helper data. Finally, the XOR module XORs the information sequence with the hash values to create the cryptographic key.

DSC Decoder

The DSC decoder is a straight forward implementation of inverse algorithm to Algorithm 1. The distance value in the helper data sets the counter value for a loop and when the break condition is reached, the PUF response is forwarded, or inverted and forwarded, depending on the corresponding inversion bit.

Viterbi Decoder

Our hard decision Viterbi implementation is based on the architecture presented in [46]. It is optimized for area, to run in FPGAs. The decoder is split in three parts: Data Path, Controller and SRAM.

- Controller: Contains the finite state machine that controls all internal operations (data-path control signals and RAM addresses) and the receiving and transmitting of encoded/decoded bits

- Data-path: All operations regarding the Viterbi algorithm are performed in the data-path.

- SRAM: Stores the current metric and the current decoded bits for each state. Each address represents the data for one state. The state with the minimal metric also contains a valid decoded message.

We chose to use dedicated block RAM as it is a widely available resource on FPGAs. The Viterbi algorithm was widely used in telecommunications over the last decades, so there exist Viterbi decoder implementations for various design goals and resource constraints.

SPONGENT Hash Function

SPONGENT [47] is a lightweight hash function based on the PRESENT S-Box [33] with a sponge construction. Aiming for low complexity, we chose SPONGENT-88/80/8 as the version with the smallest data-path and internal state. The hardware implementation mainly contains of S-Boxes, a permutation layer and a linear feedback shift register. We optimized our design such that the internal state is only stored once and used an architecture with parallel S-Boxes to avoid large multiplexers.

XOR Module

The XOR module reads in the key and XORs it with the hash values. The helper data exceeds the key size, so multiple hash values are created and XORed over the key in several iterations. Using multiple hash values permits to use a hash function with an internal state smaller than the key size. This reduces the size of the hash function. The internal state is stored in distributed RAM.

Implementation Evaluation

To be able to compare our results with existing implementations, we also used the soft decision SRAM PUF scenario by Maes *et al.* [16] with average bit error probability 15%. Goal is to reproduce the key with an mean error probability of 10^{-6}. The scenario provides a fair comparison, since [17], [18] and also DSC obtain reliability information once

	PUF Output Bits	Helper Data Bits	Slices	Block RAM Bits	Clock Cycles
Code-Offset Golay [15]	$3,696$	$3,824$	$\geq\ 907$	0	$> 24,024$
Code-Offset RM-GMC [17]	$1,536$	$13,952$	237	$32,768$	$10,298$
C-IBS RM-GMC [18]	$2,304$	$9,216$	250	0	–
DSC Conv. Code Mean Error	$1,224$	$2,176$	262	$11,264$	$30,846$

Table 1: FPGA implementations of reproduction procedures of the DSC and reference implementations synthesized for Xilinx Spartan 3E FPGAs

	DSC Dec	Viterbi Dec	SPONGENT	XOR Mod	Entire Module
Slices	18	91	78	61	262
Registers	14	32	109	44	209
LUTs	31	154	146	109	464
Block RAM Bits	–	$11,264$	–	–	$11,264$

Table 2: Synthesis results of the submodules of the DSC reproduction procedure implementation for Xilinx Spartan-3E FPGA (xc3s1200e-5fg320)

	DSC Dec	Viterbi Dec	SPONGENT	XOR Mod	Entire Module
Slices	9	50	15	23	94
Registers	12	32	21	37	111
LUTs	20	115	34	69	261
Block Rams	–	4	11	–	15

Table 3: Synthesis results of the submodules of the DSC reproduction procedure implementation for Xilinx Spartan-6 FPGA (xc6slx45-3fgg484)

during the helper data generation and only the binary PUF response during reproduction.

For the DSC implementation, we set the maximal distance between two PUF outputs to 128, so $l = 7$, which leads to a yield $y > 99.9\%$ according to Equation 5.

The results in Table 1 show that our implementation is more efficient than all comparable references in terms of number of PUF output bits ($\approx 20\%$ less) and helper data bits (43% less). The Block RAM consumption is low and the number of slices is only slightly over the best previous work.

The number of clock cycles is 20% over slower comparable work and about $2\times$ slower as the fastest implementation.

Our hardware implementation demonstrates that DSC is not only efficient in its performance. It can also be brought into practice.

Detailed synthesis results for Spartan 3E and Spartan 6 FPGAs are provided in Tables 2 and 3. The modules were synthesized with Xilinx ISE 14.6, 64 bit. We provide results for Spartan-3E (xc3s1200e-5fg320) and Spartan-6 (xc6slx45-3fgg484) FPGAs to ensure compatibility to previous results and also facilitate comparisons with future work. Note that the top level module also contains control logic, so the overall size is larger than the sum of its components.

11. CONCLUSIONS

Secure and efficient error correction is a prerequisite for using PUFs in practice. In this work, we addressed this issue with improvements in both error correction stages.

We introduced DSC, a new and efficient syndrome coding scheme for PUFs. DSC permits to give lower bounds for the reliability of the system and increases its efficiency to previous work. Our theoretical analysis demonstrated the performance of DSC and proved its security.

Further, we are the first to show that convolutional codes are a good choice as second stage error correction for PUFs. They have powerful error correction properties and we provide an area optimized implementation of a Viterbi decoder.

Bounding techniques instantaneously give worst case values for the bit error probabilities without time and resource consuming simulations.

Our performance comparison demonstrates that DSC outperforms the referenced schemes in a benchmark scenario by than 20% in terms of PUF outputs and over 40% in helper data size.

Therefore, this paper demonstrated that avoiding fixed block sizes leads to a significant cost reduction for PUF based systems.

12. ACKNOWLEDGEMENTS

We thank Dominik Merli for the discussions and Xiaoqing Wan for her work at the beginning of the project. We also thank the anonymous reviewers for their helpful feedback. This work was partly funded by the German Federal Ministry of Education and Research in the project ARAMiS through grant number 01IS11035Y.

13. REFERENCES

[1] D. E. Holcomb, W. P. Burleson, and K. Fu, "Power-up SRAM state as an identifying fingerprint and source of true random numbers," *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1198–1210, 2009.

[2] J. Petit, C. Bösch, M. Feiri, and F. Kargl, "On the potential of PUF for pseudonym generation in vehicular networks," in *IEEE Vehicular Networking Conference (VNC)*, 2012, pp. 94–100.

[3] K. Kursawe, A.-R. Sadeghi, D. Schellekens, B. Skoric, and P. Tuyls, "Reconfigurable physical unclonable

functions - enabling technology for tamper-resistant storage," in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2009, pp. 22–29.

[4] J. Guajardo, S. S. Kumar, G. J. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, P. Paillier and I. Verbauwhede, Eds., vol. 4727. Springer, Heidelberg, 2007, pp. 63–80.

[5] R. Nithyanand and J. Solis, "A theoretical analysis: Physical unclonable functions and the software protection problem," in *International Workshop on Trustworthy Embedded Devices (TrustED)*, 2012, pp. 1–11.

[6] V. van der Leest and P. Tuyls, "Anti-counterfeiting with hardware intrinsic security," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2013, pp. 1137–1142.

[7] T. Esbach, W. Fumy, O. Kulikovska, D. Merli, D. Schuster, and F. Stumpf, "A new security architecture for smartcards utilizing PUFs," in *Information Security Solutions Europe (ISSE)*, 2012.

[8] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Techniques for design and implementation of secure reconfigurable PUFs," *ACM Transactions on Reconfigurable Technology Systems (TRETS)*, vol. 2, no. 1, pp. 1–33, 2009.

[9] R. Maes and I. Verbauwhede, *Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions*. Springer, Heidelberg, 2010, pp. 3–37.

[10] U. Rührmair, S. Devadas, and F. Koushanfar, *Security based on physical unclonability and disorder*. Springer, New York Inc., 2011.

[11] R. Maes, "Physically unclonable functions: Constructions, properties and applications," Dissertation, Katholieke Universiteit Leuven, 2012.

[12] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," in *Advances in Cryptology (EUROCRYPT)*, ser. LNCS, C. Cachin and J. L. Camenisch, Eds., vol. 3027. Springer, Heidelberg, 2004, pp. 523–540.

[13] M.-D. Yu and S. Devadas, "Secure and robust error correction for physical unclonable functions," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 48–65, 2010.

[14] M. Bossert, *Channel Coding for Telecommunications*. New York: John Wiley & Sons, 1999.

[15] C. Bösch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls, "Efficient helper data key extractor on FPGAs," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, E. Oswald and P. Rohatgi, Eds., vol. 5154. Springer, Heidelberg, 2008, pp. 181–197.

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

[17] ——, "Low-overhead implementation of a soft decision helper data algorithm for SRAM PUFs," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, C. Clavier and K. Gaj, Eds. Springer, Heidelberg, 2009, pp. 332–347.

[18] M. Hiller, D. Merli, F. Stumpf, and G. Sigl, "Complementary IBS: Application specific error correction for PUFs," in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2012, pp. 1–6.

[19] R. Maes, A. Van Herrewege, and I. Verbauwhede, "PUFKY: A fully functional PUF-based cryptographic key generator," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, E. Prouff and P. Schaumont, Eds., vol. 7428. Springer, Heidelberg, 2012, pp. 302–319.

[20] V. van der Leest, B. Preneel, and E. van der Sluis, "Soft decision error correction for compact memory-based PUFs using a single enrollment," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, E. Prouff and P. Schaumont, Eds., vol. 7428. Springer, Heidelberg, 2012, pp. 268–282.

[21] M.-D. Yu, R. Sowell, A. Singh, D. M'Raihi, and S. Devadas, "Performance metrics and empirical results of a PUF cryptographic key generation ASIC," in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2012, pp. 108–115.

[22] D. J. Costello Jr. and G. D. Forney Jr., "Channel coding: The road to channel capacity," *Proceedings of the IEEE*, vol. 95, pp. 1150–1177, 2007.

[23] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[24] A. Juels and M. Wattenberg, "A fuzzy commitment scheme," in *ACM Conference on Computer and Communications Security (CCS)*. ACM, 1999, pp. 28–36.

[25] G. E. Suh, "AEGIS : A single-chip secure processor," Dissertation, Massachusetts Institute of Technology, 2005.

[26] C. Bösch, "Efficient fuzzy extractors for reconfigurable hardware," Master's Thesis, Ruhr-University Bochum, 2008.

[27] G. Schnabl and M. Bossert, "Soft-decision decoding of Reed–Muller codes as generalized multiple concatenated codes," *IEEE Transactions on Information Theory.*, vol. 41, no. 1, pp. 304–308, 1995.

[28] A. Van Herrewege and I. Verbauwhede, "Tiny application-specific programmable processor for BCH decoding," in *IEEE International Symposium on System on Chip (SoC)*, 2012, pp. 1–4.

[29] M.-D. Yu, D. M'Raihi, S. Devadas, and I. Verbauwhede, "Security and reliability properties of syndrome coding techniques used in puf key generation," in *GOMACTech Conference*, 2013, pp. 1–4.

[30] M. Majzoobi, M. Rostami, F. Koushanfar, D. S. Wallach, and S. Devadas, "Slender PUF protocol: A lightweight, robust, and secure authentication by substring matching," in *International Workshop on*

Trustworthy Embedded Devices (TrustED), 2012, pp. 33–44.

[31] A. van Herrewege, S. Katzenbeisser, R. Maes, R. Peeters, A.-R. Sadeghi, I. Verbauwhede, and C. Wachsmann, "Reverse fuzzy extractors: Enabling lightweight mutual authentication for PUF-enabled RFIDs," in *Financial Cryptography and Data Security (FC)*, ser. LNCS, A. D. Keromytis, Ed., vol. 7397. Springer, Heidelberg, 2012, pp. 374–389.

[32] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. John Wiley & Sons, 2006.

[33] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An ultra-lightweight block cipher," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, P. Paillier and I. Verbauwhede, Eds., vol. 4727. Springer, Heidelberg, 2007, pp. 450–466.

[34] F. Armknecht, R. Maes, A.-R. Sadeghi, B. Sunar, and P. Tuyls, "Memory leakage-resilient encryption based on physically unclonable functions," in *Advances in Cryptology (ASIACRYPT)*, ser. LNCS, M. Matsui, Ed., vol. 5912. Springer Berlin Heidelberg, 2009, pp. 685–702.

[35] M. Hiller, G. Sigl, and M. Pehl, "A new model for estimating bit error probabilities of ring-oscillator PUFs," in *International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. IEEE, 2013.

[36] M. Hofer and C. Boehm, "An alternative to error correction for SRAM-like PUFs," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, S. Mangard and F.-X. Standaert, Eds., vol. 6225. Springer, Heidelberg, 2010, pp. 335–350.

[37] D. Holcomb, A. Rahmati, M. Salajegheh, W. P. Burleson, and K. Fu, "DRV-fingerprinting: Using data retention voltage of SRAM cells for chip identification," in *Workshop on RFID Security and Privacy (RFIDSec)*, 2012.

[38] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *ACM/IEEE Design Automation Conference (DAC)*, 2007, pp. 9–14.

[39] R. Maes, "An accurate probabilistic reliability model for silicon PUFs," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, G. Bertoni and J.-S. Coron, Eds. Springer, Heidelberg, 2013, pp. 73–89.

[40] M. Hiller, "Optimized fuzzy extractor for PUFs on FPGAs," Diplomarbeit, Ulm University, 2011.

[41] S. Katzenbeisser, U. Kocabas, V. Rozic, A.-R. Sadeghi, I. Verbauwhede, and C. Wachsmann, "PUFs: Myth, fact or busted? a security evaluation of physically unclonable functions (PUFs) cast in silicon," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, E. Prouff and P. Schaumont, Eds., vol. 7428. Springer, Heidelberg, 2012, pp. 283–301.

[42] D. Merli, D. Schuster, F. Stumpf, and G. Sigl, "Side-channel analysis of PUFs and fuzzy extractors," in *International Conference on Trust and Trustworthy Computing (TRUST)*, ser. LNCS, J. M. McCune, B. Balacheff, A. Perrig, A.-R. Sadeghi, A. Sasse, and Y. Beres, Eds., vol. 6740. Springer, 2011, pp. 33–47.

[43] X. Boyen, Y. Dodis, J. Katz, R. Ostrovsky, and A. Smith, "Secure remote authentication using biometric data," in *Advances in Cryptology (EUROCRYPT)*, ser. LNCS, R. Cramer, Ed., vol. 3494. Springer, Heidelberg, 2005, pp. 147–163.

[44] R. Cramer, Y. Dodis, S. Fehr, C. Padro, and D. Wichs, "Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors," in *Advances in Cryptology (EUROCRYPT)*, ser. LNCS, N. Smart, Ed., vol. 4965. Springer, Heidelberg, 2008, pp. 471–488.

[45] J. Conan, "The weight spectra of some short low-rate convolutional codes," *IEEE Transactions on Communications*, vol. 32, no. 9, pp. 1050–1053, 1984.

[46] A. Chang, O. Salehi-Abari, and S. S. Woo, "Viterbi decoder," Project Report, Massachusetts Institute of Technology, 2011.

[47] A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varici, and I. Verbauwhede, "SPONGENT: A lightweight hash function," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, B. Preneel and T. Takagi, Eds., vol. 6917. Springer, Heidelberg, 2011, pp. 312–325.