When the Decoder Has to Look Twice: Glitching a PUF Error Correction

Jonas Ruchti, Michael Gruber and Michael Pehl

Chair of Security in Information Technology
Technical University of Munich, Germany
{j.ruchti|m.gruber|m.pehl}@tum.de

Abstract. Physical Unclonable Functions (PUFs) have been increasingly used as an alternative to non-volatile memory for the storage of cryptographic secrets. Research on side channel and fault attacks with the goal of extracting these secrets has begun to gain interest but no fault injection attack targeting the necessary error correction within a PUF device has been shown so far. This work demonstrates one such attack on a hardware fuzzy commitment scheme implementation and thus shows a new potential attack threat existing in current PUF key storage systems. After presenting evidence for the overall viability of the profiled attack by performing it on an FPGA implementation, countermeasures are analysed: we discuss the efficacy of hashing helper data with the PUF-derived key to prevent the attack as well as codeword masking, a countermeasure effective against a side channel attack. The analysis shows the limits of these approaches. In particular, it demonstrates the criticality of timing in codeword masking by confirming the attack's effectiveness on ostensibly protected hardware.

Keywords: physical unclonable function · fuzzy commitment scheme · fault attack · clock glitch · masking

1 Introduction

Suppose you find yourself in the shoes of a vendor needing to protect a device's firmware against unauthorised copying and modification. As your product does not have a protected non-volatile memory (NVM) suitable for the secure storage of an encryption key, you turn your attention to Physical Unclonable Function (PUF) key storage schemes. PUFs have gained much attention for similar applications in the last two decades as they can sidestep the problems of storing a secret in NVM.

By exploiting unavoidable tolerances of the manufacturing process, a PUF provides a device-unique secret. These variations are measured with a PUF circuit, such as SRAM cells [HBF07], ring oscillators (ROs) [Gas+02; SD07; YD10], or concurrent delay chains like in Arbiter PUFs [Gas+04]. In any case, the PUF circuit measurement under different challenges or of PUF circuits in different positions in a device results in a set of noisy PUF responses, which—in case of key storage systems—are then error-corrected to arrive at a sufficiently stable secret.

One big benefit of a PUF-based key storage system is that the secret generated from a PUF is only made available on the chip on demand. As a consequence, countermeasures such as tampering sensors can focus on protecting the time window in which the secret is derived and processed. The existence of invasive attacks such as the ones presented in [Hel+13; Mer+11a] shows that such countermeasures are needed. However, sensors are hardly able to detect non-invasive attacks and a variety of possible attacks have thus to be considered in the PUF context.

State of the art regarding attacks on PUFs. Attacks on PUFs encompass a large variety of different attack vectors. The likely most popular attacks are related to machine learning, e.g. [Rüh+10; Bec15; Gan+16]. Attacks in this domain mostly focus on the challenge-response behaviour of a PUFs and are therefore not of relevance when storing a secret key with a PUF where the response is usually not available from outside of the chip. Even though few works have shown that PUFs with challenge-response behaviour are also vulnerable through exploiting public helper data needed to enable error correction in the system [BWG15; SFP21], such attacks are not critical for the majority of key storage schemes today which only use single-challenge PUFs.

Another class of attacks hinges on observing the PUF measurement through side channels, including invasive attacks exploiting the photon emission of SRAM cells and Arbiter PUFs [Hel+13; Taj+14] as well as attacks using localised electromagnetic measurements of RO PUFs [Mer+11a; SF20]. The latter are not limited to invasive attacks; successful side-channel attacks on the TERO PUF [TPI19] and on the Loop-PUF [TDP20] show that even non-invasive attacks are feasible and have to be taken into account through some protection mechanism when implementing a PUF system.

However, the actual PUF is not the only potential attack target in a PUF-based key storage system. Similar to any cryptographic algorithm processing a secret, the algorithm deriving a noise-free key from a noisy PUF response is subject to hardware-related attacks. For example, the two-metric helper data scheme can enable the derivation of response bits from side-channel measurements [Teb+21]. In addition, the error correction code (ECC) decoder circuit itself can also be subject to side channel attacks [Mer+11b; MSS13; TPS17].

While the feasibility of side-channel attacks (SCAs) on PUFs has already been proven, Fault Injection Analysis (FIA) of a PUF-based key storage systems has been mostly out of scope for the community. Only few works like the fault attacks on RO- and Arbiter-based PUF primitives [Taj+15; DV14] investigated the feasibility of such attacks.

Yet, none of the existing works focused on attacks on the PUF error correction. The focus of this work lies on, the feasibility of Fault Injection Analysis of the error correction code of PUF-based key storage systems, which has so far not been explored.

Attacker Model. For the fault attacks discussed in this work, we assume an attacker who is in possession of the device under attack and can therefore tamper with the device. Consistent with state of the art, we assume that the helper data used to enable error correction is public. This is reasonable as it has to be generated per device and—since a PUF is used—no protected NVM can be expected. We further assume that the attacker can manipulate the helper data and run (possibly destructive) profiling on a set of devices of the same kind as the device under attack. Finally, we assume that the attacker can trigger the reconstruction of the key from the PUF under fault influence on the device an arbitrary number of times and distinguish whether the correct or a wrong key is derived, e.g. through observing if the device operates as expected.

Contributions. This work is focused on one concrete implementation of a PUF-based key storage system. Nevertheless, our conclusions are applicable to a more general scope. This work's contributions are:

- This work introduces a *theoretical model for a FIA on an error correcting scheme.*
- It *demonstrates the practical feasibility of the FIA* using a code concatenation of a $(7, 1, 3)$ repetition code and a $(127, 64, 10)$ Bose–Chaudhuri–Hocquenghem (BCH) code[1] implemented on an field-programmable gate array (FPGA).

[1] The code parameters are taken from [MSS13] to allow for a better comparability of the impact of the attack.

- It *discusses the applicability of two possible countermeasures*, namely of codeword masking and helper data hashing.
- It *demonstrates the limitations of the used masking schemes in hindering FIA*.
- It also *demonstrates the impact of different guessing strategies* for the unprotected case as well as given protection through codeword masking.

The rest of this work is structured as follows. After preliminaries, i.e. PUF-based key storage systems and fault attacks, are introduced in Section 2, the attack itself is described in detail in Section 3. Section 4 justifies the hardware set-up used for validating the attack experimentally, after which Section 5 presents the experiment results. After the results' implications have been discussed in Section 6, this work ends with a conclusion and an outlook in Section 7.

2 Preliminaries

Before describing the actual fault attack, the fundamentals of the underlying system are defined. This section summarises the *fuzzy commitment scheme* used for the PUF-based key storage as well as fault attacks with a focus on glitch-based attacks.

2.1 PUF-based key storage

Since a PUF's response is subject to noise and environmental effects, typically a helper data scheme involving an error correction code is used to reliably store a secret. In this work, we focus on the fuzzy commitment [JW99] helper data scheme. Figure 1 depicts a sketch of the resulting system, which is based on two phases: the *enrolment* phase and the *reconstruction* phase.

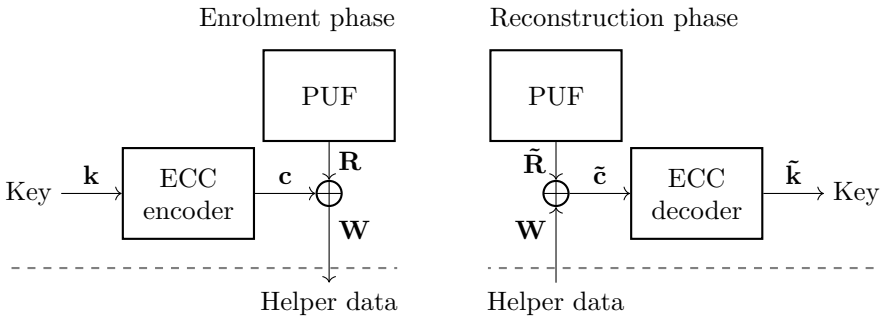


Figure 1: Fuzzy commitment scheme

During the one-time enrolment, the key to be stored $\mathbf{k}$ is encoded to a codeword $\mathbf{c}$. XORing $\mathbf{c}$ with a reference measurement of the PUF response $\mathbf{R}$ yields the so called *helper data* $\mathbf{W}$, which is then stored for later usage.

When the secret $\mathbf{k}$ is needed at a later point in time, it is reconstructed from helper data and PUF response. This process begins with a PUF measurement $\tilde{\mathbf{R}}$. As this measurement differs from the reference $\mathbf{R}$ due to noise and environmental effects, its combination with the helper data, $\tilde{\mathbf{c}}$ is also not exactly the same as the codeword calculated during the enrolment phase. However, the error in $\tilde{\mathbf{c}}$ is compensated by the system's ECC, deriving a key $\tilde{\mathbf{k}}$ which is correct with high probability $\Pr[\tilde{\mathbf{k}} = \mathbf{k}]$.

All values above the dashed line in Figure 1 are secrets and only exist within the device during its operation. The helper data $\mathbf{W}$, on the other hand, contains no information about the secrets and can be stored in a publicly accessible manner.

To extract the secrets from the system, the attack described in this work manipulates the transmission of the ECC codeword $\tilde{\mathbf{c}}$ during a reconstruction phase. By introducing faults during this transmission and observing the system's behaviour, information about $\tilde{\mathbf{c}}$ and thus $\tilde{\mathbf{R}}$ is recovered. Consequently, the attack is also applicable to other helper data schemes which process PUF and helper data in a comparable manner, like it is the case for the *code offset* construction scheme [Dod+08].

2.2 Glitch-based Fault Injection Analysis

Fault Injection Analysis is the generic term for a class of physical attacks as introduced by Boneh et al. in their seminal work [BDL00]. The underlying principle of these attacks is the deliberate violation of a device's specifications to introduce erroneous behaviour. A low-cost way to cause a violation of the critical path is glitching [Bar+06; Exi14], either using a voltage drop or a temporary increase of the clock frequency. Both effect a violation of the set-up time requirement $t_\mathrm{p} + t_\mathrm{su} < T$ [Sap06], by either raising the propagation time t_p or lowering the clock period T so a critical path's output signal is no longer stable in a register's set-up time window t_su.

Several physical glitching setups have been proposed [OC15; Kud+18; SMC20]. In this work we will focus on on-chip clock glitching, which enables a high temporal precision by inserting a precisely timed additional clock edge within a regular clock cycle [BGV11].

2.3 Notation

Upright bold-face variables denote bit vectors, as they are used within the device under attack for storage and transmission of messages and secrets. A_i is the i-th bit of the vector $\mathbf{A}$ and can either be `0` or `1`. The bits are defined to be numbered from left to right, in their order of transmission, i.e. $\tilde{\mathrm{c}}_0$ will be the first codeword bit to be transmitted to the ECC decoder and $\tilde{\mathrm{c}}_{n-1}$ the last. $\mathbf{e}_i = [\mathtt{0}, \ldots, \mathtt{1}, \ldots, \mathtt{0}]$ denotes the bit vector which is `1` at the position i and `0` elsewhere.

Important constants for the secret recovery algorithms outlined later in this section are the parameters of the ECC, which are often written as a triplet (n, k, t). n is the codeword length, which coincides with the length of the PUF secret, while k is the length of the encoded secret. t denotes the number of bit errors the ECC is guaranteed to recover from.

3 The proposed attack

This section first sketches the attack on a PUF key storage system and describes the fault model used in this work. Thereafter, the process of the attack and all necessary algorithms are described in detail.

3.1 Fault model

On a fundamental level, the reconstruction of the key from a PUF and in particular the error correction is carried out by *sequential logic*, whose memory is provided by registers capturing their inputs at a clock line's rising edge.

A possible effect of a set-up time violation is that the register stores the state of its input *before* the transition. Figure 2 shows this in an exaggerated fashion by adding a clock glitch during the propagation time of the previous clock cycle's signal.

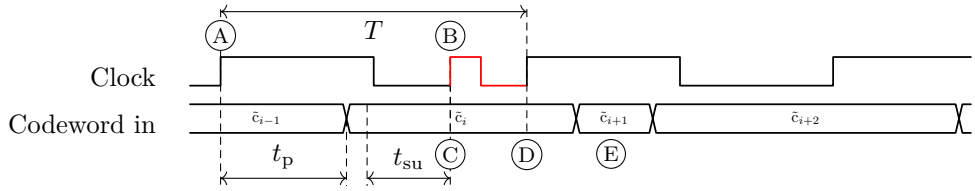


Figure 2: Codeword transmission as received by the decoder, exhibiting exemplary fault effect. The clock glitch is highlighted in red.

Evidently, in the example in Figure 2 the time between the first rising clock edge (A) and the glitch's rising clock edge (B) is sufficiently long, the codeword is stable for a sufficient time and set-up time is not violated when sampling codeword bit $\tilde{c}_i$ at time point (C). However, the glitch is too close to the following rising clock edge and the driving signal cannot propagate to the decoder's input quickly enough—the i-th codeword bit is captured again at time point (D). The value $\tilde{c}_{i+1}$ arriving at the decoder input is only available for a short time (E) as it is quickly replaced by $\tilde{c}_{i+2}$, which began propagating to the decoder at the rising clock edge (D) after the glitch. Effectively, the decoder samples $\tilde{c}_i$ twice and skips $\tilde{c}_{i+1}$.

3.2 Attack sketch

To justify the relevance of the attack, we outline an exemplary system where we consider a microcontroller device with application code stored in unprotected NVM.

The manufacturer wants to prevent unauthorised copying, and modification of the memory contents even for an attacker with physical access to the device and thus encrypts them using a device-unique key. For the key storage, a PUF system as sketched in Figure 1 is employed. Required helper data for the PUF system is considered public and stored together with the encrypted application code, where it can be read and modified by the attacker as per attacker model.

During boot-up, the device reconstructs the secret key from a PUF measurement and the helper data and uses it to decrypt the memory contents. Because all secrets only exist during runtime, tamper protection measures have to be active only as long as the device is powered, which allows the attacker to modify the hardware in a powered-down state in order to introduce a clock glitch later.

The attacker now manipulates the helper data in such a way that the error correcting code under attack is at its correction limit, i.e. such that the output key is still correctly derived but is influenced as soon as a fault injection changes a single codeword bit. Then they apply power to the device and introduce a clock glitch while the codeword is transferred to the ECC decoder. Per the previous section's fault model, this allows them to replace one codeword bit by the preceding bit's value[2].

By observing the outcome of the reconstruction phase (i.e. pass or fail) after inserting a glitch, the attacker can then reason about the two bits' difference. If the device still boots, the recovered key was unaffected by the bit replacement and both codeword bits can be concluded to be the same. If the device fails to boot, replacing the targeted codeword bit with its predecessor evidently changed the codeword and with the ECC at its error correction limit also the key.

The attacker repeats this experiment, targeting different codeword bits through repeatedly power-cycling the device, modifying the helper data, and introducing clock glitches. This way, they finally recover all bit differences of the ECC codeword and thus

[2]This work assumes a bit-serial transmission. The attack is also adaptable to larger bus widths, in a straight-forward way up to the ECC's error correction capability and even further with additional effort.

the PUF secret.

3.3 Secret extraction algorithms

The following describes required procedures to extract PUF generated secrets from a device using the previously described mechanisms. For brevity's sake, all algorithms in this section assume a perfectly reproducible glitch effect and no PUF measurement noise or environmental variation, i.e. $\tilde{\mathbf{R}} \equiv \mathbf{R}$ and $\tilde{\mathbf{c}} \equiv \mathbf{c}$, which makes all interactions with the device under attack deterministic. This assumption does not limit the applicability, since PUF measurement noise or independent extraction errors can be compensated by averaging multiple codeword extractions[3].

To represent an interaction with the device under attack, the algorithms below use a place-holder function Experiment($\mathbf{W}'[, g]$): using the (modified) helper data word $\mathbf{W}'$ and optionally a glitch position g, a reconstruction phase is carried out on the target. After a usage of the reconstructed key, Experiment returns whether or not the reconstructed key matches the key programmed during enrolment of the PUF system. Consequently, Experiment($\mathbf{W}', g$), which introduces a clock glitch during the transmission of bit g, returns false if replacing $\tilde{\mathrm{c}}_{g+1}$ with $\tilde{\mathrm{c}}_g$ leads to $\tilde{\mathbf{k}} \neq \mathbf{k}$.

Helper data manipulation. According to our fault model, introducing a clock glitch at position g data-dependently influences the codeword bit at position $g+1$. For this change to be observable, the ECC decoder needs to be at its *error correction limit*. In general, an ECC can recover from more than t bit errors in some cases, which makes the necessary helper data manipulation dependent on the codeword and glitch position. To bring the ECC to its correction limit, an attacker can invert bit $g+1$, successively add more bit flips until Experiment always fails, and then revert the modification of $g+1$. The special structure of the BCH code in the experiments, however, allows to add exactly t bit flips within the first k bits of the codeword to bring the decoder to its error correction limit, which simplifies the problem.

Algorithm 1 constructs such a helper data manipulation vector, intelligently placing the bit flips to the attacker's advantage. As only the symbol part of the codeword is modified, the improvements do not hold for redundancy bits, which will become apparent later. However, we accept this compromise to allow for an easier choice of bit flips.

Algorithm 1 Construct a n-bit vector $\mathbf{f}$ which can be used to bring the (n, k, t)-decoder to its error-correction limit, given a glitch position g and the target hamming weight t.

```
1: procedure Correction Limit(g, t)
2:     N ← {i : 0 ≤ i < k ∧ 0 ≤ |i − g| < t/2}        ▷ Define a set of positions near the glitch
3:     f_i ← 0   ∀i ∈ [0, ..., k)                      ▷ Initialise the bit flip vector f to all-zeros
4:     Choose f_i uniform randomly from {0, 1}   ∀i ∈ N \ {g + 1}
5:     while HW(f) < t do                              ▷ Increase hamming weight to t
6:         f_i ← 1 with i random from [0, k) \ N
7:     return f                                        ▷ Return bit flip vector
```

As a first step, bit positions 'near' the glitch position are chosen randomly independent uniform in line 4, ensuring that there are at most t bit flips. Choosing f_g in particular at random has an advantage, because a helper data bit flip at position g, changing the codeword bit before the glitch, inverts the outcome of Experiment($\mathbf{W}', g$). If, for some glitch position, the fault effect is not data-dependent, its behaviour will then become apparent during the attack: for such positions the outcome of Experiment is static, i.e.

[3] We assume a bit error probabilities of below 50 %. Bit error probabilities above 50 % correspond to a PUF response offset and can be compensated by flipping the corresponding helper data bit.

always failing or always succeeding, while it would be expected to differ depending on the choice of f_g in case of a data dependency.

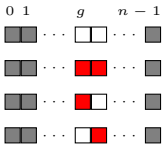
Randomising a range of helper data bits also helps to counteract the effects of glitch position jitter: as the codeword bits neighbouring the glitch position are now unbiased and independent of the codeword, measurement noise caused by imprecisely placed glitches is independent as well and can be compensated by averaging multiple trials.

Finally, the loop beginning with line 5 ensures the correct hamming weight of $\mathbf{f}$ with additional bit flips at random positions. Bit positions from the set of neighbours $\mathcal{N}$ are excluded here to preserve the previously sampled uniform distribution.

Profiling. For a successful attack, the best parameters, e.g. for alignment and timing of the clock glitch, need to be determined first. To estimate the exploitable data dependency, given a parametrisation θ, Algorithm 2 carries out four fault injections, modifying two adjacent helper data bits in all four bit patterns.

Algorithm 2 Determine a fitness measure of a point θ in the parameter space at a glitch position g, using the original helper data $\mathbf{W}$.

1: **procedure** FITNESS($\mathbf{W}, g, \theta$)
2: Pick x, y at random from $[0, n) \setminus \{g, g+1\}$ such that $x \neq y$
3: $\mathbf{f} \leftarrow$ CORRECTION LIMIT($g, t-2$), such that $\mathrm{f}_i = 0\ \forall i \in \{x, y\}$
4: $\mathbf{W}' \leftarrow \mathbf{W} \oplus \mathbf{f}$
5: $r \leftarrow 0$
6: **if** EXPERIMENT($\mathbf{W}' \oplus \mathbf{e}_x \oplus \mathbf{e}_y, g, \theta$) fails **then** $r \leftarrow r + \frac{1}{2}$
7: **if** EXPERIMENT($\mathbf{W}' \oplus \mathbf{e}_g \oplus \mathbf{e}_{g+1}, g, \theta$) fails **then** $r \leftarrow r + \frac{1}{2}$
8: **if** EXPERIMENT($\mathbf{W}' \oplus \mathbf{e}_g \oplus \mathbf{e}_y, g, \theta$) fails **then** $r \leftarrow r - \frac{1}{2}$
9: **if** EXPERIMENT($\mathbf{W}' \oplus \mathbf{e}_x \oplus \mathbf{e}_{g+1}, g, \theta$) fails **then** $r \leftarrow r - \frac{1}{2}$
10: **return** $|r|$



Two of these experiments will return the same result if an exploitable data dependency is present: regardless of the actual codeword, two experiments will have a bit difference and two will have the same bit before and at the glitch position. In this case, r will accumulate an absolute value of 1. If the results of EXPERIMENT do not depend on whether a bit difference at the glitch position is inserted via the helper data or not, the contributions to r will cancel out.

Averaging multiple calls to FITNESS thus provides an estimate of the observable data dependency as a value between 0 and 1. An attacker can use this information for a numeric optimisation of the parameter point. In the simplest case, they evaluate FITNESS averages for random glitch positions over a grid in the parameter space and then pick the optimal θ.

Codeword extraction. Having ensured that the ECC is at its correction limit, Algorithm 3 extracts the codeword by iterating through all bit positions and placing a glitch before each in turn, observing the result of the fault injection. The first codeword bit is extracted based on the assumption that the state of the data line before the transmission $\hat{\mathrm{c}}_{-1} = \mathtt{0}$, i.e. $\hat{\mathrm{c}}_0$ is $\mathtt{1}$ if the first glitch experiments with the clock glitch inserted before the transmission of bit 0 leads to a reconstruction failure.

The helper data modification of the codeword bit before the glitch position also has to be accounted for. Lines 6 and 8 invert the recovered bit if W_g had been flipped.

This algorithm only attempts to extract each bit once. To compensate measurement noise and glitch position jitter, it is sensible to run multiple trials of ATTACK on the same device and then use a majority vote on the extracted codeword bit differences.

Data error correction. Due to measurement noise or other imperfections, a perfect codeword extraction might not be possible in a real-world scenario. Since the attacker

Algorithm 3 Recover the n-bit codeword $\mathbf{c}$ assuming a set-up time violation glitch effect model, given the original helper data $\mathbf{W}$.

```
1: procedure ATTACK(W)
2:     ĉ_{-1} ← 0                          ▷ Assumption: Data line is 0 before the transmission
3:     for g ← −1 ... n − 2 do
4:         f ← CORRECTION LIMIT(g, t)      ▷ Find suitable helper data bit flip vector
5:         if EXPERIMENT(W ⊕ f, g) fails then
6:             ĉ_{g+1} ← not ĉ_g ⊕ f_g     ▷ Consecutive bits differ
7:         else
8:             ĉ_{g+1} ← ĉ_g ⊕ f_g         ▷ Current bit is the same as the last one
9:     return [ĉ_0, ..., ĉ_{n−1}]
```

has knowledge of the system's inner construction and thus knows the system's error correcting code, they can use it to recover from some bit extraction errors. In the following, ENCODE and DECODE denote an encoding and error-correcting decoding operation using an equivalent implementation of the code used in the system under attack.

Since the extraction procedure operates on bit differences instead of the codeword bits directly, we define a vector of codeword bit differences $\mathbf{d}$,

$$\mathrm{d}_i := \mathrm{c}_{i-1} \oplus \mathrm{c}_i \quad \text{for } 0 \leq i < n, \tag{1}$$

where c_{-1} is the state of the data line before the transmission of the first codeword bit c_0, which we assume to be $\mathtt{0}$ for now; if this information is unavailable, d_0 contains the attacker's guess of the first codeword bit instead. A vector $\hat{\mathbf{d}}$ for the attacker's extracted values is similarly defined using $\hat{\mathbf{c}}$.

One might be tempted to say that up to t wrongly recovered bits can be recovered by employing the ECC directly, because the original codeword $\mathbf{c}$ is a valid codeword after all. However, this is not necessarily possible in the general case: a wrong bit in $\hat{\mathbf{d}}$ compared to $\mathbf{d}$ corresponds to bit errors in $\hat{\mathbf{c}}$ from its position onward, often far more than a single bit flip. Therefore, even a single wrong bit in $\hat{\mathbf{d}}$ might not be correctable.

Still, a number of errors in $\hat{\mathbf{d}}$ can be corrected, depending on the qualities of the employed error-correction code. An important code class, to which also the BCH code used in this work belongs to, are *cyclic codes*, a subset of *linear codes*. In these, not only every linear combination of two codewords but also every cyclic shift of a codeword is a valid codeword, too [Bla03].

Note that the construction of $\mathbf{d}$ in Equation (1) almost makes it a cyclic codeword: if d_0 were defined as $\mathrm{c}_{n-1} \oplus \mathrm{c}_0$, $\mathbf{d}$ would be a linear combination of $\mathbf{c}$ and a cyclic shift of $\mathbf{c}$ and thus a valid codeword. In our case, however, we can think of $\mathbf{d}$ as a codeword with one possible bit error in position 0 (which occurs if $\mathrm{c}_{n-1} = \mathtt{1}$).

A simple procedure making use of this property is presented as Algorithm 4, which uses DECODE and ENCODE directly on the vector of extracted codeword bit differences. To compensate for the possible error due to the 'imperfect' cyclic codeword, it attempts the error correction on two variants, with and without bit 0 flipped and returns the variant where the error correction changed fewer positions.

A flipped bit 0 in $\hat{\mathbf{d}}$ corresponds to an inversion of $\hat{\mathbf{c}}$. If the all-ones word is part of the code, $\hat{\mathbf{c}}$ is just as valid a codeword as its inverted counterpart and an extraction error at position 0 is thus not detectable. If we assume a bit difference extraction error probability below $\frac{1}{2}$, a correct extraction of bit 0 is more likely than an extraction error. Thus, the original value of bit 0 is restored with the flip in line 7.

In case of a code with even minimum distance, the modification of bit 0 cannot erroneously move the codeword to a point closer to the wrong reconstruction; Algorithm 4 can thus reliably correct t errors apart from any error in position 0. For an odd minimum distance $d = 2t + 1$, this cannot be guaranteed and the bit difference error correction capability drops to $t - 1$ bits. In the case of the BCH code used in the experiments,

Algorithm 4 Error-correct a word $\hat{\mathbf{d}}$ of extracted codeword bit differences for a cyclic code.

procedure CORRECT DIFFERENCES($\hat{\mathbf{d}}$)
  $\hat{\mathbf{d}}_0 \leftarrow \hat{\mathbf{d}}, \quad \hat{\mathbf{d}}_1 \leftarrow \hat{\mathbf{d}} \oplus \mathbf{e}_0$ ▷ Copy to $\hat{\mathbf{d}}_0$, invert bit 0 for $\hat{\mathbf{d}}_1$
  $\hat{\mathbf{d}}'_i \leftarrow \text{ENCODE}(\text{DECODE}(\hat{\mathbf{d}}_i)) \quad \forall i \in \{0, 1\}$ ▷ Error-correct both variants
  if $\text{HD}(\hat{\mathbf{d}}'_0, \hat{\mathbf{d}}_0) \leq \text{HD}(\hat{\mathbf{d}}'_1, \hat{\mathbf{d}}_1)$ **then** ▷ Pick the variant with fewer errors
    return $\hat{\mathbf{d}}'_0$
  else
    return $\hat{\mathbf{d}}'_1 \oplus \mathbf{e}_0$

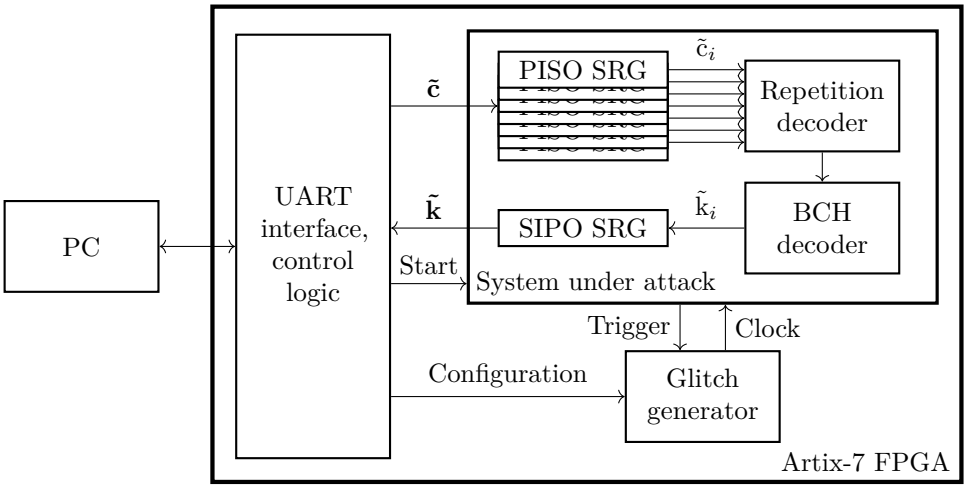


Figure 3: Simplified block diagram of the experiment set-up.

Algorithm 4 was found to reliably correct t errors after bit 0 despite the odd minimum distance d, because $d > 2t + 1$.

4 Experimental set-up

Since the attack is mainly concerned with the serial codeword transmission between PUF and ECC decoder, no complete key storage system is implemented for the experiments. In particular, the PUF is replaced with a model and the derived key is not used in a cryptographic application. This section outlines the design choices behind the hardware model in terms of its scope and additional features, which facilitate a reasonably fast validation of the attack while representing a real-world system's behaviour realistically.

4.1 Basic experiment hardware

Figure 3 shows a block diagram of the experiment set-up: a Xilinx XC7A35T-1CPG236C FPGA contains both the system under attack and a clock glitch generator; all components are configured and communicated with using a UART interface. Naturally, this model carries a number of design choices and simplifications.

PUF model. As the proposed attack only needs the data transmission to the error correction code, the PUF itself lies beyond the scope of this work and its concrete implementation is not relevant. As the PUF measurement can be assumed to be done

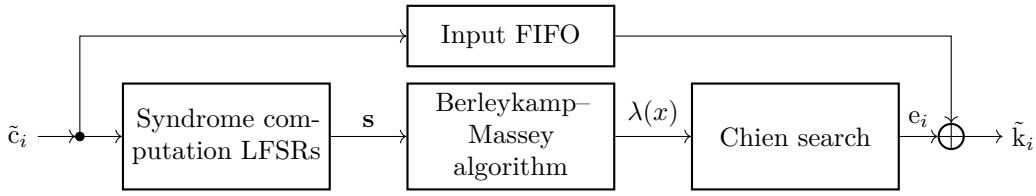


Figure 4: Block diagram of the BCH decoder hardware.

by the time the error correction begins, it is replaced with a programmable register.[4] Throughout the experiments, the PUF response was held constant, as PUF measurement noise can be compensated by averaging multiple attack runs.

Error correction code. So far, the exact code used as the PUF system's error-correction measure was not important, as long as its codeword was transmitted serially. This work's implementation closely mirrors the code used in [MSS13] and [TPS17], i.e. a concatenated code consisting of a $(7, 1, 3)$-repetition code as its inner code and a $(127, 64, 10)$-BCH code as its outer code. BCH codes have been proposed and used in the context of PUF systems a number of times [Yu+12; Kan+14], sometimes in combination with a repetition code [MVV12]. They offer good performance and efficient hardware implementations and are thus suitable for the task.

Since the PUF value is assumed to be constant and the repetition decoder consists entirely of combinational logic, it is of little importance for the functional principle of this attack. It is still included in the hardware design because its propagation delay caused by its logic has an influence on the exact timing of the system. The manipulation of one helper data bit in the attack described above corresponds to flipping one 7-bit block at the repetition decoder input.

The implementation of the BCH decoder has been generated using the software presented in [Jam97]. This code uses *systematic encoding*, i.e. the codeword is a concatenation of the 64-bit *symbol part*, which correspond to 64 key bits, and the 63-bit *redundancy part*, containing error correction information. To derive a 128 bit key two BCH code words would be used in practice.

Figure 4 presents a block diagram of its structure: linear-feedback shift registers (LFSRs) are used to compute the syndromes of the bit-serially supplied input, after which the *error locator polynomial* is determined. Based on that, the actual bit errors are calculated, which are than corrected in a stored copy of the input's first 64 bits. Since the input first-in first-out (FIFO) and the syndrome LFSRs use only a bit-wise serial input of $\tilde{\mathbf{c}}$, the described attack is directly applicable for this ECC implementation.

In this BCH implementation, the locations of the errors are determined using *Chien search*, i.e. by finding the roots of a polynomial of degree t [Jam97]. Since this polynomial has at most t roots, *exactly* t bit errors can be corrected if all errors are within the symbol part of the codeword. Since we assume a constant PUF secret, which is the same for enrolment and reconstruction, this allows for a simplification of the attack: to bring the decoder to its error correction limit, Algorithm 1, which inserts exactly t bit flips, can be used instead of a more generic helper data modification scheme.

Fuzzy commitment scheme implementation. Only the reconstruction phase is implemented, as the enrolment phase is out of the attacker's control and not relevant to the

[4] Profiling and experiment results (cf. Section 5) for different configurations agree well with the fault model. Together with simulations explaining particular behaviours (cf. Section 5.1) they substantiate that indeed the targeted ECC decoder is attacked and not just the PUF model.

attack. Thus, only the error correction based on simulated PUF secret and helper data is necessary. The usage of the reconstructed key is simulated by a comparison to a stored copy, yielding the pass/fail result.

Because the BCH has a bit-serial input, parallel-in serial-out (PISO) shift registers (SRGs) are used to convert from the parallel codeword to the serial decoder input, one for each repetition decoder input. These registers might also be present in a real-world implementation as part of a FIFO buffer to interface between the slow PUF and fast ECC decoder. Similarly, a serial-in parallel-out (SIPO) SRG is used to convert the reconstructed key to a parallel format.

On-chip glitching. In contrast to a real-world attacker, who would use an external glitch generator, on-chip glitching guarantees perfect glitch alignment. However, with a intelligent helper data modification scheme, glitch position jitter can be compensated with averaging (cf. Section 3.3). The architecture, source code, and performance of the glitch generator, which is based on the ChipWhisperer [OC15], are available as supplementary material [Mat].

4.2 Masking implementations

Masking is a well-known countermeasure against SCAs [Cha+99; RP10; GMK16]. By adding a random mask to a secret intermediate value, which is later removed again, masking effectively makes it useless to the attacker without knowledge of the ephemeral mask. The principles of masking have also been used to provide protection against Statistical Ineffective Fault Analysis (SIFA) under the assumption of a SIFA-1 fault model [Sah+20], which assumes an alternation of parts of the shares.

In the context of PUFs, masking has already been found successful against an SCA on a system similar to the one under consideration in this work [MSS13]. We adapt this codeword masking scheme in order to analyse its effectiveness against FIA. The masking scheme generates a random codeword of the ECC from some random seed and XORs it to the noisy codeword from the PUF in order to mask the decoding procedure. The mask is removed after decoding by XORing the random seed to the decoder output, which is possible due to linearity of the error-correcting code.

On a hardware implementation of a fuzzy-commitment-based PUF key storage system, masking of the decoder is nearly free in terms of required resources as the random number generator (RNG) and ECC encoder are likely already present for the enrolment phase and only an intermediate storage for the masking key and some control logic need to be added.

Masking architecture. The ECC encoder was generated using the same software as the decoder to ensure a matching code. For the RNG, a 64 bit LFSR was instantiated using a polynomial from [Alf96]. Note that this is by no means a cryptographically secure RNG, which could be exploited in a more advanced attack. As the RNG's potential weaknesses are not the focus of this work, it is merely important that its output is (approximately) bias-free. To achieve this, the LFSR is left free-running and sampled once for each reconstruction phase. During the experiments, the codeword bits are attacked in random order to ensure any periodicity effects the LFSR might show cannot affect the results.

Two slightly different approaches are analysed in this work, shown in Figure 5. In the first one, (a), the random mask is applied to the BCH decoder's input. This scheme might suffice to protect against side-channel attacks targeting the BCH decoder, as the attacks target the decoder's input FIFO, whose contents are now randomised.

A second, more complete variant, (b), applies the mask before the repetition decoder, thus masking the complete concatenated code. This implementation has the disadvantage

of needing one XOR gate for each repetition decoder input and thus comes with a slightly higher hardware overhead.

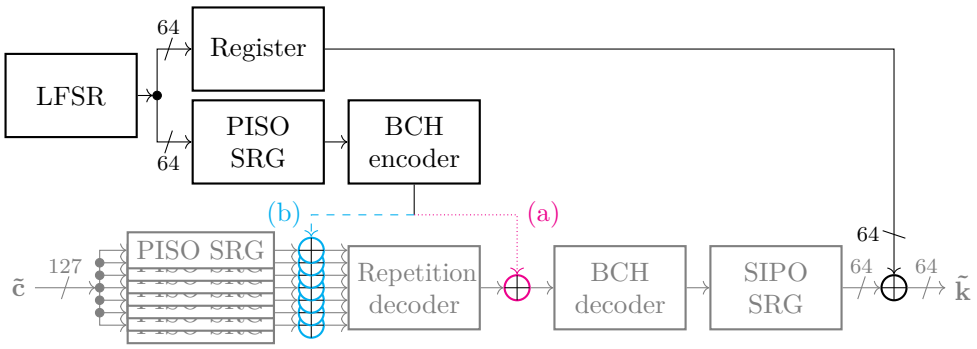


Figure 5: Masking block diagram with control and clock signals omitted for brevity and the already present reconstruction phase circuit drawn in grey. Two masking implementations are tested: the codeword mask is either added (a) before the BCH decoder or (b) before the repetition decoder.

4.3 Experiment procedure

For a representative evaluation, experiments were carried out on 15 FPGA boards. For all tested implementation variants, an attack procedure based on two phases was carried out independently for each board:

- **Profiling.** Before a codeword was extracted, the optimal glitch parameters were determined using Algorithm 2. To match a realistic scenario, where an attacker cannot choose or change the system's codeword, as closely as possible, a single random codeword per FPGA board was used[5]. To limit operator bias, the maximum was found using a peak search on FITNESS evaluations of uniformly random parameters, which required a comparatively high number of 250 000 samples. An attacker can employ a guided search or pick the timings manually, requiring considerably fewer data points. Results of the profiling step, available as supplementary material, provide additional support to our fault model.
- **Attack.** Using the per-FPGA optimal glitch timings, the attack was carried out using Algorithm 3. 250 trials of this algorithms were used for 100 random attacked codewords per FPGA. To monitor the attack as it progressed, the extracted codeword bit differences were computed on-line based on the average of the current trials.

4.4 Attack success metrics

After extracting a secret codeword from a device, the number of bit extraction errors gives a first indication for the attack's success. However, since the position of any extraction errors is unknown to the attacker, they need, in general, to guess more than this number of bits to reach the correct secret. This section discusses different metrics for bit guessing after the attack, used during the experiments to assess the attack's power.

Since the attacker can only extract bit differences between subsequent bits, it is sensible to judge their success based on the number of correct bit differences. In the following, 'bit extraction errors' refer to errors in the bit *differences* of codeword and, respectively, key.

[5] A cross-check repeating the experiments on a subset of the FPGAs with different codewords did not reveal any dependence on the particular codeword.

Residual guess entropy (RGE). Lacking any further information, a sensible approach for an attacker would be to guess codewords based on their error count, i.e. the attacker would try all codewords with one bit flip respective to their extracted value, then two additional bit flips, and so on. An upper bound of the number of bits the attacker needs to guess to find x bit errors in an l-bit word is the max-entropy:

$$\mathrm{RGE}(x, l) = \log_2 \left(\sum_{i=0}^{x} \binom{l}{i} \right). \tag{2}$$

If the system under attack uses systematic encoding, i.e. the key bits are available directly as a subset of the codeword bits, the attacker can try to only extract these key bits. If x bit extraction errors were made during that process, the residual guess entropy for the key-only attack becomes

$$\mathrm{RGE}_{\hat{\mathbf{k}}}(x) := \mathrm{RGE}(k, x). \tag{3}$$

As previously discussed, if a cyclic code is used, its decoder can be used by an attacker to error-correct their extracted codeword. For simplicity, we assume that the attacker will always guess bit 0 due to its special role and will be able to correct t bit extraction errors among the remaining codeword bits. The RGE thus becomes

$$\mathrm{RGE}_{\hat{\mathbf{c}}}(x_{1+}) := \begin{cases} 1 & \text{for } x_{1+} \leq t \\ 1 + \mathrm{RGE}(n-1, x_{1+} - t) & \text{otherwise} \end{cases}, \tag{4}$$

where x_{1+} is the number of bit extraction errors for the codeword bits 1 to $n-1$.

Note that either strategy can be better. For low extraction error counts, a significant part can be error-corrected if the complete codeword is extracted, whereas $\mathrm{RGE}_{\hat{\mathbf{c}}} > \mathrm{RGE}_{\hat{\mathbf{k}}}$ for higher error counts, since the attacker has to find the errors within $n > k$ bit positions. For example, for the $(127, 64, 10)$-code used for the experiments, extracting the whole codeword leads to a lower residual guess entropy only if there are less than 16 bit extraction errors (assuming an equal distribution of errors within the codeword).

Smart guessing strategies. If additional information about the system is known, an attacker can guess bits more intelligently. We consider two approaches:

Maximum-variance (MV) guessing. As multiple fault injection experiments are carried out for each codeword bit to compensate for measurement noise by averaging, estimating the measurement variance per bit is possible. This variance intuitively maps to a confidence in the extracted bit and an attacker can try to guess bits in order of decreasing measurement variance.

This metric is computed as the number of bits, as ordered by their measurement variance, which need to be adapted for all extraction errors to be compensated or until the remaining errors can be corrected using the ECC. As with the residual guess entropy, bit 0 is always adapted first in the case of a codeword extraction.

Maximum error probability (ME) guessing. In some cases, an attacker might be able to profile the attack more extensively or in other ways obtain information at which positions a secret extraction is less likely succeed. They would then adapt the bit positions with the highest extraction error probability first.

In the experiments, this metric is calculated a-posteriori, using the collected data from all boards to estimate all bit positions' extraction error probabilities. The number of bits to be flipped to reach a correct key/codeword is then determined analogously to the MV guess count.

5 Experiment results

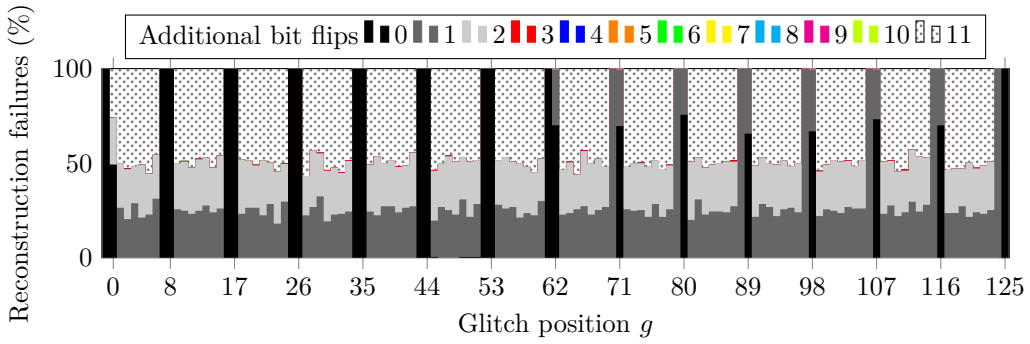
Using the procedure outlined in Section 4.3, 15 FPGA boards were used to carry out 100 attacks on randomly chosen secret keys each. This section presents the results of each implementation variant's 1 500 attack experiment results, highlighting four boards showing representative behaviour.

5.1 Unprotected implementation

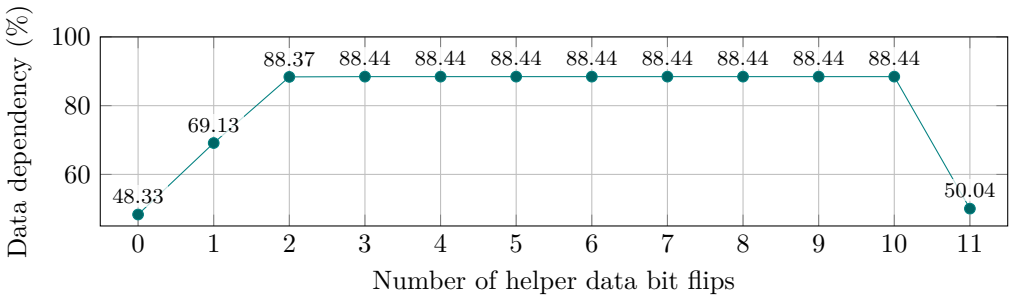
Before examining the proposed countermeasure, we first demonstrate the attack's feasibility on an unprotected implementation. This section begins with a short analysis of the clock glitch's influence on the system before proceeding with the actual attack results.

Error correction limit under glitch influence. Following the argumentation of Section 3.1, we would expect a clock glitch to have no effect at all for fewer than t bit flips in the (fault-less) codeword because the decoder can always recover from a single additional error. However, reconstruction failures were already observed for much fewer helper data bit flips.

To analyse this behaviour, 250 random 64-bit keys were encoded and the effects of a glitch at each codeword bit position was recorded for different number of helper data bit flips. The previously determined optimum glitch timing was used and the 0 to 11 helper data bit flips' positions were chosen at random within the k symbol bits of the codeword, excluding the glitch position g and $g-1$.



(a) Share of reconstruction failures when introducing a clock glitch at a specific position.



(b) Observed data dependency of the fault injection results.

Figure 6: Fault injection behaviour depending on the number of inserted helper data bit flips, based on experiments with 250 random codewords.

Figure 6a shows the share of the recorded reconstruction failures for each glitch position; the number of additional bit flips at which the reconstructions start to fail is indicated by

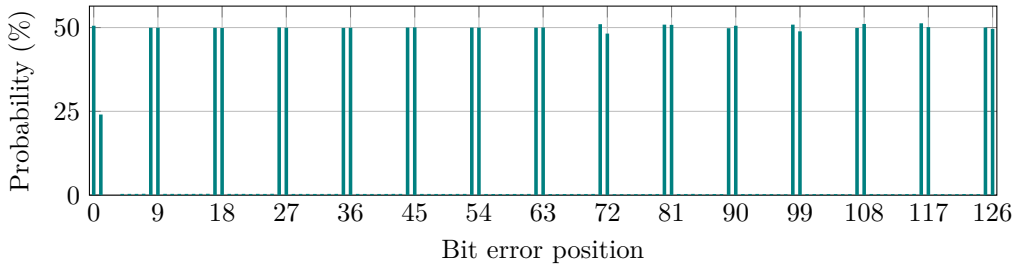


Figure 7: Bit extraction error probability over the bit position, estimated from all experiments.

the bars' colours. As expected, all reconstructions fail at $t + 1$ bit errors, since the flip positions do not permit a compensation by the clock glitch.

However, reconstructions start to fail much earlier than at t bit flips. First, a number of glitch positions, visible as regularly spaced black vertical bars, lead to a reconstruction failure in every case, even without any helper data manipulation. These glitch positions thus cannot exhibit any useful data dependency[6]. Since they coincide with two control signals with 9-bit period, it is likely that a glitch at these positions disturbs the decoder's internal control logic, affecting the reconstructed key.

Second, even with one bit of helper data manipulation, a significant share of clock-glitched reconstructions begins to fail. Even more so, with the exception of a few cases at three bit flips (drawn in red), the experiment's outcomes do not change from two to ten bit flips and Figure 6a remains mostly colourless. This is more directly visible in Figure 6b, where the share of pass/fail results in line with the set-up time violation model is shown depending on the number of additionally inserted bit flips. In simulations, a similar behaviour occurred when some syndrome LFSRs were left unaffected by the clock glitch, which also fits the intuition: as soon as the syndrome computation units become desynchronised, the error correction capability suffers.

However, since a behaviour like this cannot be presumed from a general system under attack, the attacks in the remainder of this section are carried out as they were described earlier, with helper data manipulation bringing the error-correcting code to its error correction limit before the insertion of clock glitches. Since the observable data dependency, as Figure 6b shows, is not worse for this case, this approach does not degrade the attack's performance. Further implications of the evidently exploitable data dependency for fewer artificially introduced bit flips are discussed later, in Section 6.

Attack results. As expected, the bit positions with no or limited data dependency highlighted in Figure 6a and discussed in the previous section also appear during the attack as bit positions with high extraction error probability. Figure 7 shows the indeterminable bits (i.e. with 50 % error probability) with their regular 9-bit spacing.

Apart from these positions, Figure 7 only has a very small 'error floor', indicating that the attack performs well with respect to measurement noise. This is corroborated by the attack's progress on the number of extraction errors within a codeword over the number of trials in Figure 8a, which is mostly constant despite a growing number of averages.

To find the locations of enough of the on average 14.9 bit errors to arrive at a correctable codeword, the maximum-variance strategy is well-suited for a majority of the cases. Because the indeterminable bits always result in a reconstruction failure, the compensation of the uniformly chosen helper data bit flip immediately before the glitch position results in

[6] A glitch timing optimisation specific to these glitch positions could not reveal any beneficial timings, either. Thus, data extraction with the proposed method seems to be impossible for these bits.

Table 1: Result statistics after 250 trials for different FPGA board subsets.

Statistic/metric	Board(s)	$\hat{\mathbf{c}}$ extraction			$\hat{\mathbf{k}}$ extraction		
		min.	avg.	max.	min.	avg.	max.
Bit extraction errors	best	6	14.2	18	2	7.4	11
	all	6	14.8	27	2	7.7	15
	worst	9	15.4	27	4	8.2	15
RGE (in bits)	best	1	21.5	41.3	11	30.3	39.8
	all	1	24.7	67.2	11	31.1	47.7
	worst	1	27.9	67.2	19.4	32.4	47.7
MV guesses	best	0	8.2	72	5	14.4	16
	all	0	10.4	97	5	19.7	62
	worst	0	12.7	97	11	38.8	62
ME guesses	best	0	7.9	16	6	14.3	16
	all	0	8.6	19	6	14.5	16
	worst	0	9.4	19	10	14.7	16

maximum measurement variance. As Figure 8b shows, the majority of the errors can be found by this strategy after the variances have been determined with a few averaged trials.

Figure 8b also reveals a few faint black lines in its upper half, though. Because Algorithm 1 used for the helper data modification places bit flips only within the key part of the codeword, the stuck bits in the final 63 bits of the codeword cannot be detected by high measurement variance—instead, they have almost zero variance. Consequently, the naïve MV guessing strategy cannot economically recover codewords with too many zero bit differences at always-failing bit positions within the codeword's redundancy part. Though, in the experiments, these cases merely make up 3 % of all 1 500 codewords.

If the likely error positions are known, this information can be used by means of the ME guessing strategy. As expected, the outlier codewords of Figure 8b do no longer appear in Figure 8c, dropping the worst-case guess count to 19 bit. Naturally, the attack also performs better on average, decreasing from 10.4 bit to 8.6 bit guesses.

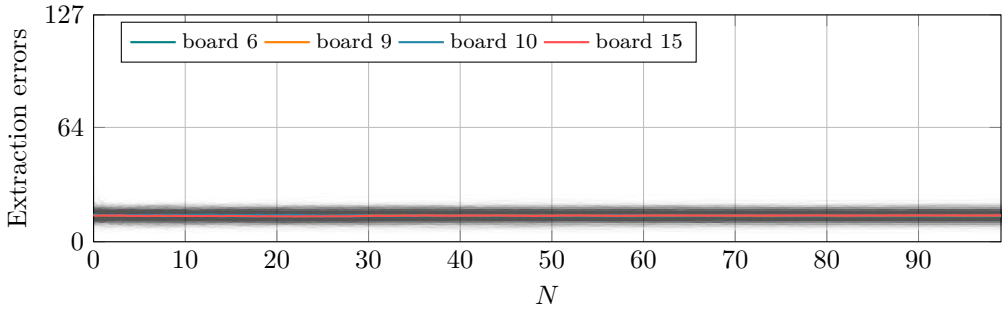
Table 1 summarises the attack's performance. There, the final number of bit errors, residual guess entropy, and guess numbers using the two strategies outlined before, are juxtaposed for an attack targeting the complete 127-bit codeword or only the 64-bit secret key. As, depending on the scenario, an attacker might depend on extracting data from a single device or could run the attack on multiple devices, it also includes statistics for the best- and worst-performing FPGA boards for each metric.

In the average case, extracting the codeword yields better results than only the key, as the average error count is below 16 bit and using the system's error correction is advantageous. Also note, again, the uneconomically high worst-case guess counts for the maximum-variance strategy, even with the best-performing hardware. However, even if more complete profiling and the ME strategy are not available, MV-guessing within the key on a range of devices could work around this issue, as the best-board worst-codeword value of 16 bit indicates.

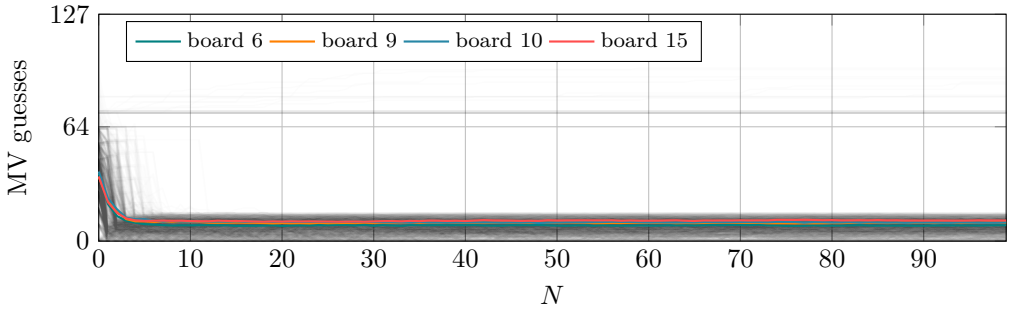
5.2 Codeword masking variants

Having shown the feasibility of the attack on an unprotected implementation, we direct our attention towards masking as a possible countermeasure. This section attempts the same attack first on a key storage system where the random mask is applied after the repetition decoder, before the BCH decoder, and second on the same system with the repetition decoder's input masked as well.

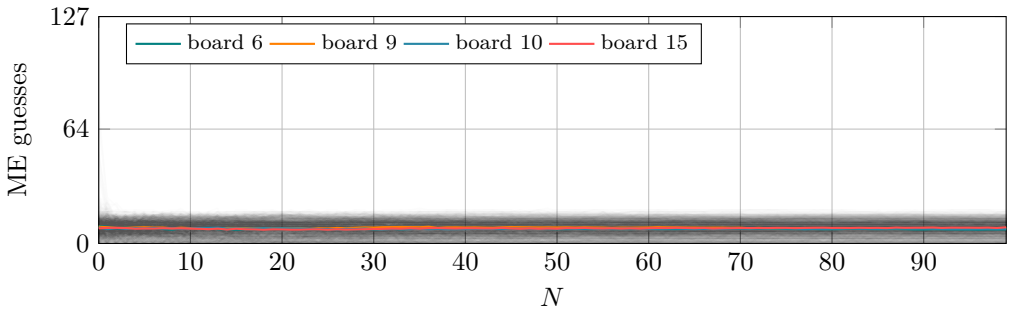
Mask applied to BCH decoder input. First, we investigate the slightly lower-cost masking variant, which applies the random mask to the BCH decoder's input.



(a) Number of extraction errors within the codeword bit differences.



(b) Number of codeword maximum-variance guesses necessary.



(c) Number of necessary codeword maximum extraction error probability guesses.

Figure 8: Progress of the attack over the first 100 trials. Values for all codewords are shown as thin lines in the background, means for FPGA boards as coloured lines.

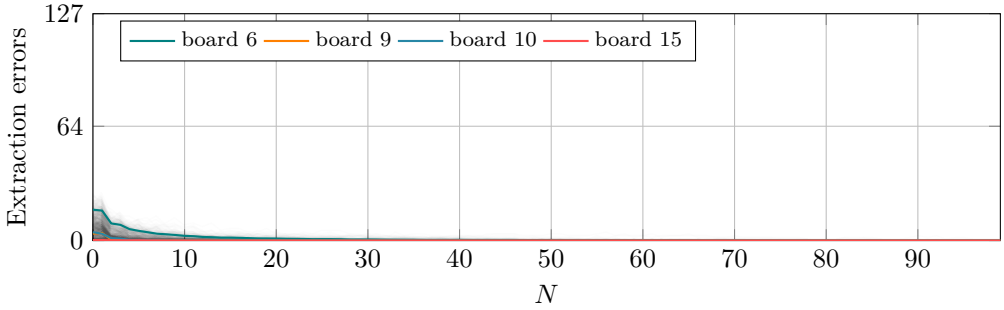


Figure 9: Attacking the configuration with the BCH decoder input masked, the extraction error count quickly converges to zero for all boards.

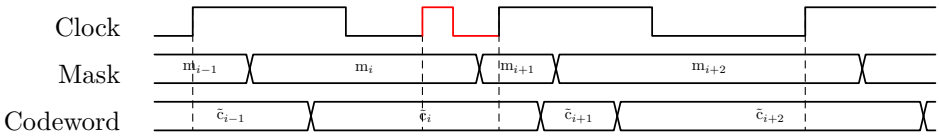


Figure 10: Waveform sketches for a set-up time violation fault attack assuming an imperfectly aligned codeword mask.

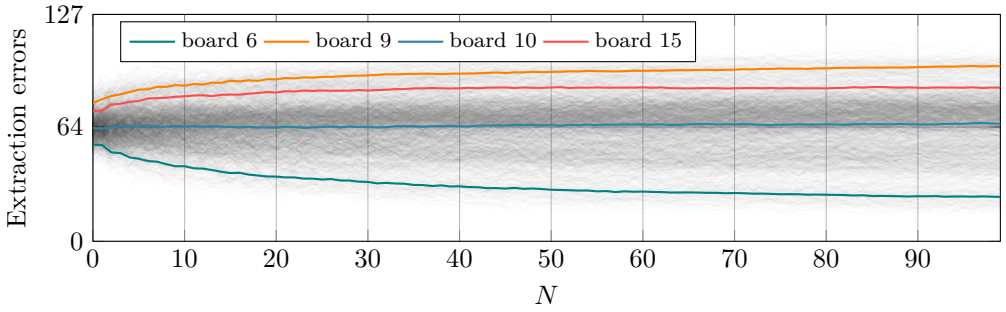


Figure 11: Attack progress over the first 100 trials on the fully masked implementation.

Since the attack targets the now-masked codeword input, we would expect it to be mitigated by this countermeasure. However, Figure 9 shows that the opposite is the case: the attack performs much better than on the unprotected implementation and even for the worst-case board 6, the extraction error count quickly converges to zero. In fact, all 1 500 tested codewords were perfectly extractable after 250 trials.

To explain this behaviour, we can take a look at the serial transmission of the secret codeword and the mask as seen by the BCH decoder's input, ignoring the XOR gate's propagation delay for now. Because the codeword has to propagate through the repetition decoder while the mask arrives directly from the BCH encoder's output register, we expect each clock cycle's codeword bit to be slightly delayed with respect to the mask bit. A clock glitch can now be inserted as shown in Figure 10: the codeword transmission is affected in the same way as for the unprotected implementation while the mask, due to its shorter propagation time, is received unaltered. Any bit difference of the masked codeword caused by the fault injection thus is only based on the secret codeword—the mask cannot hinder the attack at all.

If we also take the XOR gate's propagation delay into account, we can reason why this masked implementation is even easier to attack than the unprotected system. The XOR gate increases the secret codeword's propagation time, thus providing the attacker with more room to place a clock glitch without disturbing the decoder's internal critical paths. With the timing less critical and unwanted side effects less likely, the secret extraction then functions closer to the idealised model.

Complete masking of the concatenated decoder. Based on this reasoning, we expect masking the repetition decoder input to yield a more effective countermeasure, because both mask and codeword have now to propagate through the repetition decoder and thus have a better-matched signal delay. Indeed, the average of the attacks on board 10 shown in Figure 11 stays close to half of the codeword length—the fault injections provide the attacker close to no information about the secret codeword.

However, the countermeasure does not seem to work equally well on all FPGA boards. Board 6, for example, arrives at an average of 21.1 bit extraction errors.

Curiously, board 9's curve bends upwards, indicating an inverted data dependency

of Experiment. Note that this behaviour is distinct from an inverted codeword, i.e. a flipped extracted bit difference 0. In the case of board 9, a reconstruction failure correlates with two identical consecutive codeword bits while a success correlates with a bit change.

This apparent contradiction to the fault model can be explained if we presume a situation opposite to the previous masking variant. If the mask arrives *after* the codeword, a clock glitch can be inserted such that the codeword is transferred normally while one mask bit is replaced by its predecessor. In half of the cases, the random mask bits around the glitch position do not differ and this glitch will not have an effect. If they did differ before the replacement, the reconstruction will fail, but only if the change was not offset by a relative difference of codeword bits.

The attack results suggest that the propagation delays of mask and codeword are, on average, indeed closely matched. However, due to hardware tolerances, they differ slightly between devices. In some cases, like board 6, the mask arrives slightly earlier like with the previous implementation; in other cases, like board 9, the codeword arrives earlier at the BCH decoder. Intermediate degrees of protection, like the exemplary board 15, also occur.

An attacker can naturally also make use of the inverted data dependency. For the attack on this variant, an additional bit guess is included in the metrics, since the attacker cannot know the polarity. Table 2 summarises the results.

Table 2: Attack statistics for the fully masked concatenated decoder.

Statistic/metric	Board(s)	$\hat{\mathbf{c}}$ extraction			$\hat{\mathbf{k}}$ extraction		
		min.	avg.	max.	min.	avg.	max.
Bit extraction errors	best	11	21.1	35	2	4.9	10
	all	11	47.3	63	2	21.3	40
	worst	48	58.9	63	19	29.8	40
RGE (in bits)	best	2	51	87.4	12	23.4	38.4
	all	2	104.3	123.5	12	54.2	65
	worst	109.1	120.3	123.5	54.7	63	65
MV guesses	best	2	30.1	48	4	19	43
	all	2	78.1	116	4	51.2	65
	worst	88	103.5	116	57	63.7	65
ME guesses	best	2	46.7	103	3	36.9	65
	all	2	95.1	116	3	60.8	65
	worst	91	105.4	116	59	64	65

Because of the overall higher extraction error probabilities, extracting the key only is advantageous in this case, as the two result columns in Table 2 show. As the errors are now not mostly concentrated on a few constant positions like with the unprotected bitstream, the ME guessing strategy based on global statistics performs worse than the MV strategy operating on the current codeword's measurements only. All in all, even this countermeasure cannot be considered too effective with an average of 19 bit MV guesses on the best-attackable FPGA board.

6 Result discussion

Although the attack was only carried out on simplified model hardware, more general conclusions can be drawn from the results.

Necessity of helper data manipulation. Since the attack depends on artificially bringing the ECC decoder to its error correction limit, it can be prevented by helper data manipulation detection. If, for example, a hash of the helper data is added to the reconstructed key, any change to the helper data will immediately let the reconstruction fail[7].

[7]For further helper data modification detection schemes, we refer the reader to [Del+15].

However, the initial experiments with the unprotected implementation showed some interesting behaviour: the data dependency was apparent for a wide range of bit flips within the codeword. Because a small number of bit errors might be present due to the devices's ageing (or the attacker heating it up) anyway, this would enable the described attack even without helper data manipulation. Naturally, this would introduce some complications for the attacker:

- The codeword the attacker extracts is offset by the present bit errors, which limits the correction of additional extraction errors.
- A glitch position jitter can no longer be compensated by averaging. The helper data manipulation previously ensured that fault injection results with a small position offset lead to uncorrelated results, which is no longer the case.
- The profiling step becomes much more difficult without helper data manipulation and can also generate false positives: a seemingly optimal glitch timing might just perturb the decoder internally, yielding promising statistics without the behaviour fitting the fault effect model. Without influencing the codeword, the glitch's effect cannot be examined thoroughly.

Masking complications. The experiment results could establish the importance of matched propagation delays for the efficacy of masking as a fault attack countermeasure. This is similar to the effects of glitches in the context of side channel attacks [MPG05], though the data dependency occurs not because of an attacked gate's non-linearity but because of the observability of a fault-induced input change.

For a designer, these propagation delay asymmetries are hard to compensate. Initial experiments in compensating the skew with manually inserted look-up tables (LUTs) used as delay elements were fruitless, as the adjustment was too coarse-grained. FPGA synthesis and implementation tools do not offer help equalising delays beyond the usual limits for synchronous designs, so a fully manual routing would have been required. In application-specific integrated circuit (ASIC) design, these issues are similarly hard to avoid, as [MPG05] argues.

Even if the delays are nominally matched, hardware variations can upset the designer's plans. Despite all FPGAs using the same bitstream, delay mismatches in both directions were apparent in the last experiment, both enabling an exploitable data dependency. This might be evaded by shortening the paths as much as possible to decrease the influence of routing element propagation delay variation, which naturally adds additional strain to the design process or might not be feasible at all.

Using multiple masking codewords, i.e. with one's bits arriving before and one's after the codeword, would do away with the issues of the precise mask timing. However, requiring twice the randomness and additional logic would also significantly complicate the design.

7 Conclusion and outlook

While side channel leaks had already been investigated in the context of PUF key storage systems, this work sheds some light onto the power of FIA in this domain. In contrast to previous attacks targeting PUF primitives, it focused on the vulnerability of the error correction code required by such schemes to reliably reconstruct a secret key. Theoretical consideration showed a possible attack threat, which was then validated with practical experiments. The investigation of two countermeasures showed their limitations and the need for further research in this direction: we discussed, in theory, that hashing the helper data with the corrected PUF secret prevents the attack completely only if helper data manipulation is the only means to bring the error correcting code to its correction limit;

codeword masking can, in principle, be used to counter the attack, yet the experiments showed that the timing of the mask can be too critical for this countermeasure to be practical. Thus, further research is needed to investigate the possible attack without helper data manipulation and to develop more effective countermeasures. Nevertheless, this work is another step towards understanding threats for PUF-based key storage systems and, thus, towards increasing their security.

Acknowledgement

This work was partly funded by the Federal Ministry of Education and Research with the project APRIORI through grant number 16KIS1389K.

References

[Alf96] Peter Alfke. *Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators*. XAPP 052. Xilinx, 7th July 1996. URL: https://www.xilinx.com/support/documentation/application_notes/xapp052.pdf.

[Bar+06] Hagai Bar-El et al. 'The Sorcerer's Apprentice Guide to Fault Attacks'. In: *Proceedings of the IEEE* 94.2 (Feb. 2006), pp. 370–382. ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2005.862424. URL: http://ieeexplore.ieee.org/document/1580506/.

[BDL00] Dan Boneh, Richard A. DeMillo and Richard J. Lipton. 'On the Importance of Eliminating Errors in Cryptographic Computations'. In: *Journal of Cryptology* 14.2 (Nov. 2000), pp. 101–119. DOI: 10.1007/s001450010016.

[Bec15] Georg T. Becker. 'The Gap Between Promise and Reality: On the Insecurity of XOR Arbiter PUFs'. In: *Cryptographic Hardware and Embedded Systems – CHES 2015*. Ed. by Tim Güneysu and Helena Handschuh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 535–555. ISBN: 978-3-662-48324-4.

[BGV11] Josep Balasch, Benedikt Gierlichs and Ingrid Verbauwhede. 'An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs'. In: *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, Sept. 2011. DOI: 10.1109/fdtc.2011.9.

[Bla03] Richard E. Blahut. *Algebraic codes for data transmission*. OCLC: 76956531. Cambridge University Press, 2003. ISBN: 978-0-511-07429-5.

[BWG15] Georg T. Becker, Alexander Wild and Tim Güneysu. 'Security analysis of index-based syndrome coding for PUF-based key generation'. In: *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on*. IEEE. 2015, pp. 20–25.

[Cha+99] Suresh Chari et al. 'Towards Sound Approaches to Counteract Power-Analysis Attacks'. In: *Advances in Cryptology — CRYPTO' 99*. Ed. by Michael Wiener. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 398–412. ISBN: 978-3-540-48405-9.

[Del+15] Jeroen Delvaux et al. 'Helper Data Algorithms for PUF-Based Key Generation: Overview and Analysis'. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.6 (June 2015), pp. 889–902. ISSN: 0278-0070, 1937-4151. DOI: 10.1109/TCAD.2014.2370531. URL: http://ieeexplore.ieee.org/document/6965637/.

[Dod+08] Yevgeniy Dodis et al. 'Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data'. In: *SIAM Journal on Computing* 38.1 (Jan. 2008), pp. 97–139. DOI: 10.1137/060651380.

[DV14] Jeroen Delvaux and Ingrid Verbauwhede. 'Fault Injection Modeling Attacks on 65 nm Arbiter and RO Sum PUFs via Environmental Changes'. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 61.6 (June 2014), pp. 1701–1713. ISSN: 1549-8328, 1558-0806. DOI: 10.1109/TCSI.2013.2290845. URL: https://ieeexplore.ieee.org/document/6728716/.

[Exi14] Exide. *Glitching for n00bs*. 2014. URL: https://recon.cx/2014/slides/REcon2014-exide-Glitching_For_n00bs.pdf.

[Gan+16] Fatemeh Ganji et al. 'Strong Machine Learning Attack Against PUFs with No Mathematical Model'. In: *Proceedings of the 18th International Conference on Cryptographic Hardware and Embedded Systems — CHES 2016 - Volume 9813*. New York, NY, USA: Springer-Verlag New York, Inc., 2016, pp. 391–411. ISBN: 978-3-662-53139-6. DOI: 10.1007/978-3-662-53140-2_19. URL: https://doi.org/10.1007/978-3-662-53140-2_19.

[Gas+02] Blaise Gassend et al. 'Silicon physical random functions'. In: *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*. Washington, DC, USA: ACM, 2002, pp. 148–160. ISBN: 1-58113-612-9.

[Gas+04] Blaise Gassend et al. 'Identification and authentication of integrated circuits'. In: *Concurrency and Computation: Practice and Experience* 16.11 (2004), pp. 1077–1098.

[GMK16] Hannes Gross, Stefan Mangard and Thomas Korak. *Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order*. Cryptology ePrint Archive, Report 2016/486. https://eprint.iacr.org/2016/486. 2016.

[HBF07] Daniel E. Holcomb, Wayne P. Burleson and Kevin Fu. 'Initial SRAM State as a Fingerprint and Source of True Random Numbers for RFID Tags'. In: *Proceedings of the Conference on RFID Security*. 2007.

[Hel+13] Clemens Helfmeier et al. 'Cloning Physically Unclonable Functions'. In: *Proceedings of the IEEE Int. Symposium of Hardware-Oriented Security and Trust*. IEEE, June 2013.

[Jam97] Ernest Jamro. 'The Design of a VHDL Based Synthesis Tool for BCH Codecs'. Master Thesis. University of Huddersfield, Sept. 1997. URL: http://home.agh.edu.pl/~jamro/bch_thesis/bch_thesis.html.

[JW99] Ari Juels and Martin Wattenberg. 'A fuzzy commitment scheme'. In: *Proceedings of the 6th ACM conference on Computer and communications security - CCS '99*. the 6th ACM conference. Kent Ridge Digital Labs, Singapore: ACM Press, 1999, pp. 28–36. ISBN: 978-1-58113-148-2. DOI: 10.1145/319709.319714.

[Kan+14] Hyunho Kang et al. 'Cryptographic key generation from PUF data using efficient fuzzy extractors'. In: *16th International Conference on Advanced Communication Technology*. 2014 16th International Conference on Advanced Communication Technology (ICACT). Pyeongchang, Korea (South): Global IT Research Institute (GIRI), Feb. 2014, pp. 23–26. ISBN: 978-89-968650-3-2. DOI: 10.1109/ICACT.2014.6778915. URL: http://ieeexplore.ieee.org/document/6778915/.

[Kud+18] Christian Kudera et al. 'Design and Implementation of a Negative Voltage Fault Injection Attack Prototype'. In: *2018 IEEE International Workshop on Physical Attacks and Inspection of Electronics (PAINE)*. 2018, pp. 1–6.

[Mat] Supplementary Material. *Will be published after acceptance of the paper.*

[Mer+11a] Dominik Merli et al. 'Semi-invasive EM Attack on FPGA RO PUFs and Countermeasures'. In: *6th Workshop on Embedded Systems Security (WESS'2011)*. Taipei, Taiwan: ACM, Oct. 2011.

[Mer+11b] Dominik Merli et al. 'Side-Channel Analysis of PUFs and Fuzzy Extractors'. In: *Trust and Trustworthy Computing*. Ed. by Jonathan M. McCune et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 33–47. ISBN: 978-3-642-21599-5.

[MPG05] Stefan Mangard, Thomas Popp and Berndt M. Gammel. 'Side-Channel Leakage of Masked CMOS Gates'. In: *Topics in Cryptology – CT-RSA 2005*. Ed. by Alfred Menezes. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 351–365. ISBN: 978-3-540-30574-3.

[MSS13] Dominik Merli, Frederic Stumpf and Georg Sigl. *Protecting PUF Error Correction by Codeword Masking*. Fraunhofer AISEC, May 2013. URL: http://eprint.iacr.org/2013/334;.

[MVV12] Roel Maes, Anthony Van Herrewege and Ingrid Verbauwhede. 'PUFKY: A Fully Functional PUF-Based Cryptographic Key Generator'. In: *Cryptographic Hardware and Embedded Systems – CHES 2012*. Ed. by Emmanuel Prouff and Patrick Schaumont. Vol. 7428. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 302–319. ISBN: 978-3-642-33026-1. DOI: 10.1007/978-3-642-33027-8_18.

[OC15] Colin O'Flynn and Zhizhang Chen. 'ChipWhisperer: An OpenSource Platform for Hardware Embedded Security Research'. In: *IN: CONSTRUCTIVE SIDE-CHANNEL ANALYSIS AND SECURE DESIGN - COSADE*. 2015.

[RP10] Matthieu Rivain and Emmanuel Prouff. 'Provably Secure Higher-Order Masking of AES'. In: *Cryptographic Hardware and Embedded Systems, CHES 2010*. Ed. by Stefan Mangard and François-Xavier Standaert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 413–427. ISBN: 978-3-642-15031-9.

[Rüh+10] Ulrich Rührmair et al. 'Modeling attacks on physical unclonable functions'. In: *Proceedings of the 17th ACM conference on Computer and communications security*. CCS '10. Chicago, Illinois, USA: ACM, 2010, pp. 237–249. ISBN: 978-1-4503-0245-6. DOI: http://doi.acm.org/10.1145/1866307.1866335. URL: http://doi.acm.org/10.1145/1866307.1866335.

[Sah+20] Sayandeep Saha et al. 'A Framework to Counter Statistical Ineffective Fault Analysis of Block Ciphers Using Domain Transformation and Error Correction'. In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 1905–1919. DOI: 10.1109/TIFS.2019.2952262.

[Sap06] Sachin S. Sapatnekar. 'Static timing analysis'. In: *EDA for IC implementation, circuit design, and process technology*. CRC press, 2006, pp. 6–1.

[SD07] Gookwon Edward Suh and Srinivas Devadas. 'Physical Unclonable Functions for Device Authentication and Secret Key Generation'. In: *ACM/IEEE Design Automation Conference (DAC)*. 2007, pp. 9–14.

[SF20] Mitsuru Shiozaki and Takeshi Fujino. 'Simple electromagnetic analysis attack based on geometric leak on ASIC implementation of ring-oscillator PUF'. In: *Journal of Cryptographic Engineering* (2020), pp. 1–12.

[SFP21] Emanuele Strieder, Christoph Frisch and Michael Pehl. 'Machine Learning of Physical Unclonable Functions using Helper Data - Revealing a Pitfall in the Fuzzy Commitment Scheme'. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.2 (2021).

[SMC20] Albert Spruyt, Alyssa Milburn and Łukasz Chmielewski. 'Fault Injection as an Oscilloscope: Fault Correlation Analysis'. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.1 (Dec. 2020), pp. 192–216. DOI: 10.46586/tches.v2021.i1.192-216. URL: https://tches.iacr.org/index.php/TCHES/article/view/8732.

[Taj+14] Shahin Tajik et al. 'Physical Characterization of Arbiter PUFs'. In: *Cryptographic Hardware and Embedded Systems – CHES 2014*. Ed. by Lejla Batina and Matthew Robshaw. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 493–509. ISBN: 978-3-662-44709-3.

[Taj+15] Shahin Tajik et al. 'Laser Fault Attack on Physically Unclonable Functions'. In: *2015 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. 2015 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC). Saint Malo, France: IEEE, Sept. 2015, pp. 85–96. ISBN: 978-1-4673-7579-5. DOI: 10.1109/FDTC.2015.19. URL: http://ieeexplore.ieee.org/document/7426155/.

[TDP20] Lars Tebelmann, Jean-Luc Danger and Michael Pehl. 'Self-secured PUF: protecting the loop PUF by masking'. In: *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer. 2020, pp. 293–314.

[Teb+21] Lars Tebelmann et al. *Analysis and Protection of the Two-metric Helper Data Scheme*. Cryptology ePrint Archive, Report 2021/830. https://eprint.iacr.org/2021/830. 2021.

[TPI19] Lars Tebelmann, Michael Pehl and Vincent Immler. 'Side-Channel Analysis of the TERO PUF'. In: *Constructive Side-Channel Analysis and Secure Design*. Ed. by Ilia Polian and Marc Stöttinger. Vol. 11421. Cham: Springer International Publishing, 2019, pp. 43–60. ISBN: 978-3-030-16349-5. DOI: 10.1007/978-3-030-16350-1_4.

[TPS17] Lars Tebelmann, Michael Pehl and Georg Sigl. 'EM Side-Channel Analysis of BCH-based Error Correction for PUF-based Key Generation'. en. In: *Proceedings of the 2017 Workshop on Attacks and Solutions in Hardware Security - ASHES '17*. Dallas, Texas, USA: ACM Press, 2017, pp. 43–52. ISBN: 978-1-4503-5397-7. DOI: 10.1145/3139324.3139328. URL: http://dl.acm.org/citation.cfm?doid=3139324.3139328.

[YD10] Meng-Day (Mandel) Yu and Srinivas Devadas. 'Recombination of Physical Unclonable Functions'. In: *35th Annual GOMACTech Conference*. Reno, NV: United States. Dept. of Defense, Mar. 2010.

[Yu+12] Meng-Day Yu et al. 'Performance metrics and empirical results of a PUF cryptographic key generation ASIC'. In: *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*. 2012 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST 2012). San Francisco, CA: IEEE, June 2012, pp. 108–115. ISBN: 978-1-4673-2341-3. DOI: 10.1109/HST.2012.6224329. URL: https://ieeexplore.ieee.org/document/6224329/.