IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2024, No. 2, pp. 154–189. DOI:10.46586/tches.v2024.i2.154-189

Impeccable Keccak

Towards Fault Resilient SPHINCS+ Implementations

Ivan Gavrilan[1], Felix Oberhansl[1], Alexander Wagner[1,2], Emanuele Strieder[1,2] and Andreas Zankl[1,2]

[1] Fraunhofer Institute for Applied and Integrated Security (AISEC), Garching, Germany,
firstname.lastname@aisec.fraunhofer.de
[2] Technical University of Munich (TUM), Munich, Germany,
firstname.lastname@tum.de

Abstract. The standardization of the hash-based digital signature scheme SPHINCS$^+$ proceeds faster than initially expected. This development seems to be welcomed by practitioners who appreciate the high confidence in SPHINCS$^+$'s security assumptions and its reliance on well-known hash functions. However, the implementation security of SPHINCS$^+$ leaves many questions unanswered, due to its proneness to fault injection attacks. Previous works have shown, that even imprecise fault injections on the signature generation are sufficient for universal forgery. This led the SPHINCS$^+$ team to promote the usage of hardware countermeasures against such attacks. Since the majority of operations in SPHINCS$^+$ is dedicated to the computation of the Keccak function, we focus on its security. At the core, hardware countermeasures against fault injection attacks are almost exclusively based on redundancy. For hash functions such as Keccak, straightforward instance- or time-redundancy is expensive in terms of chip area or latency. Further, for applications that must withstand powerful fault adversaries, these simple forms of redundancy are not sufficient. To this end, we propose our impeccable Keccak design. It is based on the methodology presented in the original *Impeccable Circuits* paper by Aghaie et al. from 2018. On the way, we show potential pitfalls when designing impeccable circuits and how the concept of active security can be applied to impeccable circuits. To the best of our knowledge, we are the first to provide proofs of active security for impeccable circuits. Further, we show a novel way to implement non-linear functions without look-up tables. We use our findings to design an impeccable Keccak. Assuming an adversary with the ability to flip single bits, our design detects all attacks with three and less flipped bits. Attacks from adversaries who are able to flip four or more bits are still detected with a high probability. Thus, our design is one of the most resilient designs published so far and the only Keccak design that is provably secure within a bit-flip model. At an area overhead of factor 3.2, our design is competitive with state-of-the-art designs with less resilience.

Keywords: Keccak, fault injection, impeccable circuits, active security, SPHINCS+, post-quantum cryptography

1 Introduction

In 2023, NIST released the first draft of FIPS 205, concerning the standardization of the stateless hash-based digital signature SPHINCS$^+$ [NIS23] with SHAKE-256 [NIS15] as underlying hash function. The extendable output function SHAKE-256 is built from the Keccak primitive [BDPvA11]. Initially, the standardization of SPHINCS$^+$ was expected to move a lot slower, as it was classified as an alternative candidate at the end of the third round of the NIST PQC standardization process. However, NIST promoted it to a

Received: 2023-10-15 Accepted: 2023-12-15 Published: 2024-03-12

standardization candidate and justified this decision with the solid security assumptions of SPHINCS$^+$ and the fact that it relies on completely different assumptions than all other schemes selected for standardization [AAC+22].

While the cryptography community is confident in SPHINCS$^+$'s theoretical security, its implementation security remains a major concern. The first fault attack on the original SPHINCS scheme was presented in [CMP18]. Since then, the attack was adapted for SPHINCS$^+$ and demonstrated to be feasible for both software [GKPM18] and hardware implementations [ALCZ20]. The attack stands out due to its relaxed adversarial model. An attacker can inject a fault at numerous points in time. It is possible to target the data flow and the control flow of an implementation in various ways, e.g. it does not matter how many instructions are skipped or how many data bits are flipped. Further, an adversary does not need to be in possession of valid signatures. Within a few iterations of the fault attack, the adversary is able to forge the signature for an arbitrary message. Similar attacks on the Dilithium signature framework exist [EAB+23, GBP18], but either require more precise fault injections, the knowledge of valid and faulty signature pairs, or they can be easily mitigated by countermeasures with minimal overhead. For completeness, it should be mentioned that also fault attacks targeting Winternitz one-time signatures [WWO+23] or SHA-3 [BGS15, LAFW17] directly exist. However, since these attacks require more precise fault injections, which corresponds to a higher expenditure of time and money, they are not included in the adversarial model for most applications.

In light of its proneness to fault attacks, the SPHINCS$^+$ team recommended investigation of hardware countermeasures in their submission to the third round of the NIST PQC standardization process [ABB+22]. Genêt [Gen23] backed this reasoning by showing the limited applicability of software countermeasures. In particular, caching of intermediate values could be shown to be ineffective for SPHINCS$^+$. The author comes to the conclusion that - for software implementations - redundancy is the best option to date, even if it introduces a significant performance overhead. However, for highly precise fault injection attacks such as laser fault injection, simple redundancy might be insufficient.

1.1 State-of-the-Art

At hardware level, there are many ways to implement countermeasures efficiently and reliably. The protection of common control flow modules such as finite-state machines or bus systems is straightforward, e.g. via integrity bits or sparse encodings. Modules that implement more complex permutations on large data structures are harder to protect against physical faults.

Impeccable Circuits. The key idea of the *Impeccable Circuits* paper [AMR+20] by Aghaie et al. and its follow-up papers [SRM20, RSM21] was to protect data with concurrent, redundant processing, where the redundant data is encoded, such that an adversary needs to flip more than two bits to corrupt the circuit. While the concept behind impeccable circuits is theoretically sound and provides a thorough protection against physical faults, it has not seen much adoption for cryptographic algorithms. In [RBSBG20], the authors apply it to AES and combine it with masking to mitigate power and electromagnetic side-channel attacks. Their work demonstrates, that designers of impeccable circuits still need to put a lot of thought into the encoding, such that non-linear permutations can be implemented efficiently. For their AES design, the authors use an orthogonal encoding of the AES state. To implement all four operations within an AES round efficiently, the input of the S-boxes is the decoded output of a correction module. In [BBM+22], various implementations of impeccable circuits were evaluated with laser fault injection.

Active, Combined, and Composable Security. The resilience of circuits against fault injection adversaries is hard to measure. In [DN20], the formal notion of d^{th}-order active security was proposed [DN20], which assumes that an adversary can flip up to $d-1$ wires. It is unclear how an impeccable circuit needs to be designed, such that it meets this security notion. Current research focuses more on formal approaches to construct gadgets, i.e. re-usable components, that can be combined into d^{th}-order active secure circuits [SMG16, DN20, FGM+23, DN21, BEF+23, FRBSG22] and to verify these circuits [RBFSG22, RBRSS+21, AWMN20, NOV+22]. A different approach is to use information-theoretic MAC tags as countermeasure against fault injection [RDMB+18, DMAN+18]. These approaches are almost exclusively combined with countermeasures against passive side-channel attacks, mostly masking. Further, the approaches used to model attackers are still evaluated [RBSG23, DN22].

Heuristic Security. Independent from these formal and security-focused approaches, multiple Keccak designs with security claims purely based on heuristics were proposed. Representative of this approach, the work of Luo et al. [LLF16] is described here. Other works that also fall into this category will be used for a detailed analysis later in this paper. Luo et al. [LLF16] proposed a solution that is based on parity checks. In addition to the original Keccak function, the system contains a predictor and a compressor. The predictor uses the inputs of a function to predict parity values of the function's outputs. The compressor consumes the outputs and computes its own parity values that are compared with those from the predictor. The proposed design uses parity checks, i.e. two bit-flips might suffice to compromise the implementation. Moreover, if the input data is corrupted at a point in time where the parity check for the last function was completed and the data is not yet loaded into the next function and its predictor, a single bit-flip is sufficient.

This design was developed to protect Keccak against random errors, which is more a safety than a security issue, and fault attacks, which are only relevant for security. In the safety domain it is common practice to model faults by assuming a certain probability that bits to flip due to environmental effects. The relevant metric for such designs is the probability that a certain number of bit-flips is not detected in a sufficiently elaborate simulation. Therefore the results of heuristic analysis provide a meaningful metric for the resilience of a design. In that regard, Luo et al. derive an error coverage of 83.6 % of *stuck-at-1* and *stuck-at-0* faults. For fault attacks, however, it is common to consider the worst case scenario, namely an adversary with white-box knowledge of the design and its flaws, as well as the ability to inject precise faults. It is questionable if an error-coverage of 83.6 % for random faults is sufficient do mitigate attacks from such an adversary. As a result, the formal methodologies described above are preferred by the research community, even though the relationship with realistic fault attacks is still being researched.

1.2 Our Contributions

Since Keccak is the most frequent and time consuming operation during signature generation, its protection against fault attacks is the first step towards a secure SPHINCS+ implementation. Therefore, in this paper, we focus on the protection of the Keccak function. We briefly discuss the integration of countermeasures for the complete SPHINCS+ scheme.

We introduce the first Keccak implementation that is resistant against all faults that manipulate less than four bits. We proof the resilience of the circuit in a formal model, by showing that our design is 3^{rd}-order active secure. Moreover, this fault (or faults) must be precise, i.e. the adversary must manipulate values of determined bits, otherwise the attack will be detected. The bit-flip fault model is widely used in research [AMR+20, DN20] and was examined in practical works [BBM+22, CGV+22]. Later in this paper, we discuss

this fault model in relation to our design. In short, we can derive a strict lower limit for the number of bit-flips and probabilities for n random bit-flips to corrupt the design. We show that even for arbitrary faults in the original data successful fault injection into our design must be highly precise. This mitigates, for example, the grafting trees attack on SPHINCS$^+$ [CMP18]. Furthermore, we evaluate the real-world resilience of our design using related work on practical fault attacks and models.

In contrast to current research, we focus only on security against active fault attacks and ignore passive side-channel attacks, since they are less relevant for SPHINCS$^+$. Our solution is inspired by the impeccable circuits concept [AMR$^+$20, SRM20, RSM21]. More precisely, we build on this work and provide the following contributions:

- A generalized investigation how the concept of active security can be applied to impeccable circuits and what implementation pitfalls need to be avoided on the example of [SMG16].
- A generalized impeccable AND gate for an efficient processing of the encoded data.
- A formal proof that our impeccable AND gate is 3^{rd}-order strong-non-accumulative.
- A complete impeccable Keccak implementation exploiting the simple and efficient encoding of sub-structures without a loss of fault resilience. We include proofs that the complete permutation is 3^{rd}-order active secure.
- A detailed implementation study that demonstrates that our design can be implemented with an area overhead of factor 3.2, a comparison with state-of-the-art designs and their potential flaws, and a discussion of the practicability of the assumed fault model.
- A discussion how a complete implementation of SPHINCS$^+$ can be protected.

We publish our impeccable Keccak design and all resources we used during the security evaluation of our design.[1]

Outline. Section 2 includes information on the grafting trees attack and the Keccak function itself. In Section 3, the concepts of active security and impeccable circuits are revisited, fault attacks on impeccable circuits are formalized, and a method to efficiently prove the active security of impeccable circuits is presented. Further, a design of an impeccable AND gate is presented and its active security and non-accumulation are formally proven. Section 4 describes our encoding of the Keccak state and the design of the five impeccable Keccak steps. In Section 5 we show our implementation results, compare them with state-of-the-art designs, propose how a complete SPHINCS$^+$ implementation can be protected, and discuss the assumed fault model.

2 Background

This section covers the necessary background on the grafting tree fault attack targeting SPHINCS$^+$ and Keccak. Further, we introduce the notation we use throughout this paper.

Notation. We perform computations in the finite field $\mathbb{F}$. Most operations are done either in the binary field $\mathbb{F}_2$ or in the extended binary field $\mathbb{F}_{2^k}$. We denote matrices by bold capital letters (e.g. $\mathbf{A}$), matrix elements by capital letters with indices (e.g. $A[i,j]$ or $A(i,j)$), vectors by bold small letters (e.g. $\mathbf{a}$), vector elements by small letters with an index in brackets (e.g. $a[i]$ or $a(i)$), and single bits or variables by small letters (e.g. a).

[1] https://github.com/Fraunhofer-AISEC/impeccable-keccak

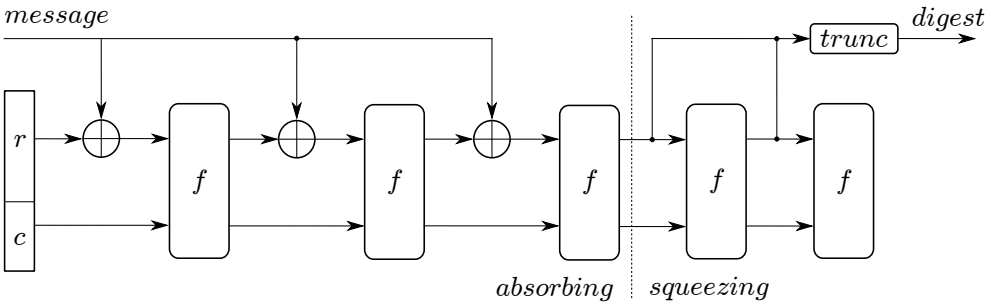


Figure 1: The KECCAK sponge function. It consists of two phases: *absorbing* —the module consumes the input *message*, *squeezing* — KECCAK produces the output *digest*. The function f represents KECCAK-$f[b]$, r denotes the rate, c is the capacity.

2.1 SPHINCS+ and the Grafting Trees Fault Attack

SPHINCS$^+$ is a stateless hash-based digital signature scheme [ABB+22]. It is based on a combination of Merkle hash trees, Winternitz one-time signatures, and the forest of random subsets few time signature scheme. In [CMP18] the authors presented the so-called *Grafting Trees* fault attack targeting the SPHINCS signing operation. This attack is not only applicable to SPHINCS but to any signature scheme based on a multi-tree architecture, i.e. Goldreich or GMSS construction [Gol87, BDK+07] and XMSS^{MT} [HRB13]. Therefore, SPHINCS$^+$ is affected as well. During the key generation of a multi-tree architecture only the top-most Merkle tree is generated. All lower Merkle trees and the respective signatures to authenticate these are only generated during a signing operation. The grafting trees fault attack exploits these recurrent generations of the same signatures. It does so by introducing a fault into the message calculation, which is the generation of the public key of one of the lower Merkle trees. Hence, the same Winternitz one-time key pair signs a tampered message. A single reuse - a one-time signature key pair signing two different messages - is sufficient to significantly degrade the security of the signature scheme [GBH18]. The fault can be introduced in numerous ways and over a long period of time, since it is only important that the message signed with the one-time signature differs from the original message. The faulted signature will verify correctly with a high probability, thus verifying after signing is not a viable countermeasure.

2.2 Keccak

KECCAK is a sponge function that is at the core of all SHA-3 hash functions. These, in turn, are fundamental building blocks of hash-based PQC algorithms such as the SPHINCS$^+$ digital signature scheme. Moreover, SHA-3 is an important building block of many other cryptographic algorithms.

The KECCAK function processes a 3-dimensional block (state) of size $b = 5\times5\times2^l$, where $l \in [0\dots6]$. Input blocks are subsequently absorbed into the state and a round function KECCAK-$f[b]$ is applied to them. KECCAK-$f[b]$ can be defined via general KECCAK-$p[b, n_r]$ function, where b is the block size and n_r is the number of rounds. Therefore, we only use the KECCAK-$f[b]$ notation in the following. For SHA-3, $n_r = 24$ and $b = 1600$. As soon as the complete message is consumed, the hash digest can be squeezed from the KECCAK state (Figure 1).

KECCAK-$f[b]$ consists of $n = 12 + 2l$ rounds:

$$\mathcal{R} = \iota \circ \chi \circ \pi \circ \rho \circ \theta, \tag{1}$$

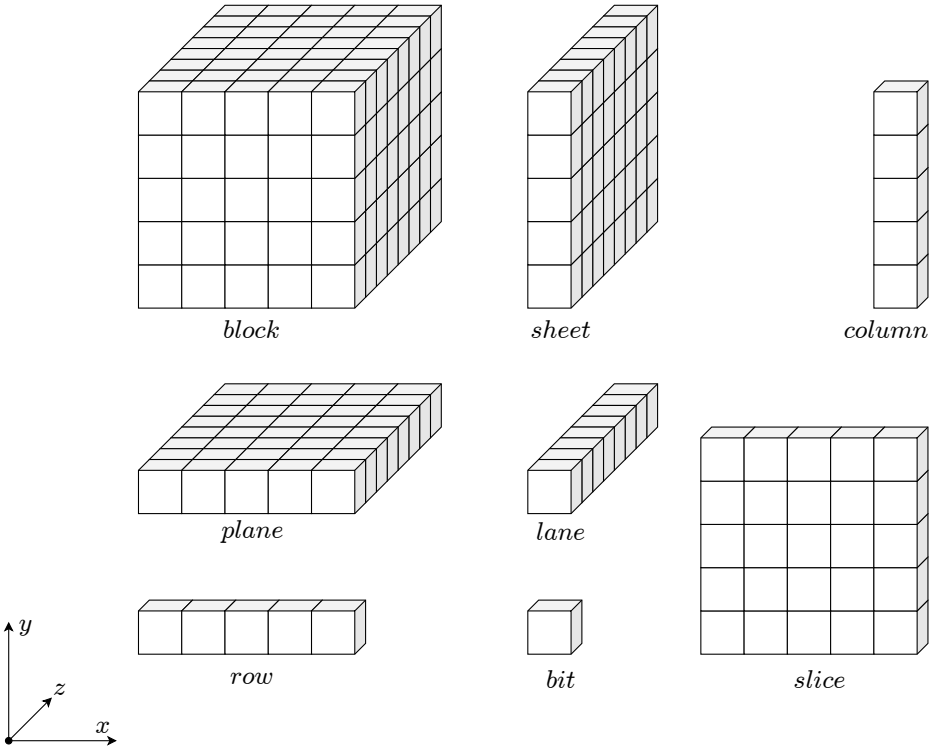


Figure 2: The KECCAK state and its substructures (according to [BDPvA11]). The complete state, also referred to as *block*, has a size of $5 \times 5 \times w$ elements, *sheet* and *plane* are of dimension $5 \times w$, a *slice* has a size of 5×5, *row* and *column* are one-dimensional arrays and consist of 5 elements, *lane* consists of w elements, the fundamental element is a single *bit*.

where θ, ρ, π, χ and ι permutations are defined as:

$$\theta : A(x,y,z) = a(x,y,z) \oplus \bigoplus_{y=0}^{4} a(x-1,y,z) \oplus \bigoplus_{y=0}^{4} a(x+1,y,z-1), \tag{2}$$

$$\rho : A(x,y,z) = a(x,y,z-c_{(x,y)}), \text{ where } c_{(x,y)} \text{ is a constant,} \tag{3}$$

$$\pi : A(x,y,z) = a(x',y',z) : (x,y) = (y', 2x' + 3y'), \tag{4}$$

$$\chi : A(x,y,z) = a(x,y,z) \oplus (\overline{a}(x+1,y,z) \wedge a(x+2,y,z)), \tag{5}$$

$$\iota : A(0,0,z) = a(0,0,z) \oplus rc(i,z), \text{ where } \mathbf{rc}(i) \text{ is the } i\text{-th round constant.} \tag{6}$$

3 Efficient and Active-secure Impeccable Circuits

This section reiterates the principles of active security and impeccable circuits as described in [DN20] and [AMR+20]. Further, we revisit the coding scheme, which has already been applied to secure PRESENT [SMG16] and AES [RBSBG20] implementations against fault attacks. Subsequently, we establish a methodology to analyze the active security of linear and non-linear impeccable circuits efficiently. According to it, we show that, although the design of linear impeccable circuits appears to be simple, it involves pitfalls that reduce their active security order. We demonstrate this on the example of [SMG16]. We

also propose a novel mechanism for the secure implementation of non-linear functions in presence of a coding scheme. This approach is used to introduce a generalized impeccable AND gate that will be proven to be active secure and strong non-accumulative.

3.1 Active Security

Fault attacks are classified as active attacks, where the attacker actively tampers with the circuit to violate its functionality or corrupt the processed data. Therefore, circuits can be classified according to their resilience against such attacks. We use the definition of active security from [DN20].

Definition 1. Order of active security (according to [DN20]). A gadget (circuit) is d^{th}-order active secure if any set of d faults on the gadget's (circuit's) intermediate variables results in either abort or a correct output.

Remark 1. In contrast to [DN20], we use d instead of k for the active security notation. We also allow the attacker to violate input and output variables.

According to Definition 1, we can define a fault.

Definition 2. Fault. A fault injected into the circuit is an unexpected modification of a single variable.

Remark 2. Variables can be represented as vectors or matrices (in the instance of Keccak e.g. as shown in Figure 2). However, the smallest representation of the variable is a bit. Therefore, in this paper, we consider variables as single bits. Further, a bit can be an intermediate value in combinatorial logic or a value stored inside a flip-flop.

According to our remarks, active security can also be expressed as follows. Assume, there exists a circuit $\mathcal{F}$ that applies a function $f(\cdot)$ to the input value x. Let $f(x)$ be vulnerable to fault injection. Considering that $f(x)$ can be calculated redundantly (either time- or instance/area- redundant), we index it as $f(x)_i$. The result of $\mathcal{F}$ is then a sequence $f(x)_1, f(x)_2 \dots f(x)_n$. If an error is present, it is detected if:

$$\forall\, i, j \in [1, n]\ \exists\, f(x)_i \neq f(x)_j \tag{7}$$

If $i = j$, the given condition is not violated. Therefore, the minimum positive number of variables that has to be modified to violate Equation (7), is the order of the active security of $\mathcal{F}$.

The authors of [DN20] proposed the notions of non-accumulation and strong non-accumulation, which help to prove the active-security of a circuit composed from smaller gadgets.

Definition 3. d-Non-Accumulative (d-NA) (according to [DN20]). A gadget (circuit) $\mathcal{F}$ is d-NA if for any set of $d' \leq d$ errors, the gadget (circuit) either aborts or gives an output with at most d' errors.

Definition 4. d-Strong Non-Accumulative (d-SNA) (according to [DN20]). A gadget (circuit) $\mathcal{F}$ is d-SNA if for any set of d_1 errors on each input and every set of d_2 errors on the intermediate values, with $d_1 + d_2 \leq d$, the gadget (circuit) either aborts or gives an output with at most d_2 errors.

A circuit that is built only from d-SNA gadgets is d-NA and therefore also d^{th}-order active secure. In this paper, we either use these composability notions from [DN20] or our own methodology to prove the active security of impeccable circuits. We show that, for impeccable circuits, the non-accumulation notions can be replaced by other security requirements (Section 3.3.1, Section 3.3.3) and how impeccable circuits can be verified efficiently for these requirements. These notions are required for impeccable circuits, since the validation of their d-NA and d-SNA properties is not always trivial. However, if an impeccable circuit is d-NA or d-SNA, the security analysis can be simplified.

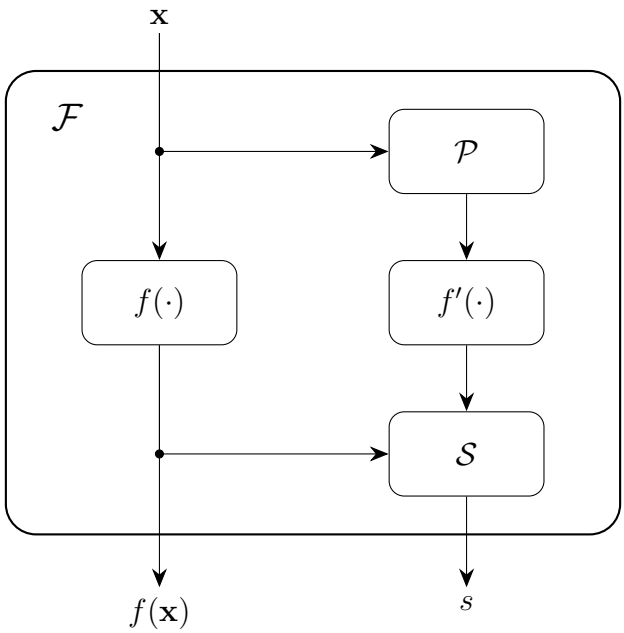


Figure 3: Structure of the module $\mathcal{F}$ according to [AMR+20]. The input vector $\mathbf{x}$ is encoded by $\mathcal{P}$ and processed by $f'(\cdot)$. The checkpoint $\mathcal{S}$ analyzes the outputs of $f(\cdot)$ and $f'(\cdot)$ and produces the signal s, which depends on the correctness of the results.

3.2 Impeccable Circuits

Aghaie et al. [AMR+20] proposed a general solution for the reliable implementation of a function $f(\cdot)$ in hardware. Its generalized structure is shown in Figure 3. The module $\mathcal{F}$ encodes the input vector $\mathbf{x} = (x_1\, x_2\, \ldots\, x_n)$ by $\mathcal{P} : \mathbf{x} \mapsto \mathcal{P}(\mathbf{x})$ and computes $f(\mathbf{x})$ and $f'(\mathcal{P}(\mathbf{x}))$, assuming $f'(\cdot)$ exists. Then let $\mathbf{y} = f(x)$ and $\mathbf{y}' = f'(\mathcal{P}(\mathbf{x}))$. A checkpoint $\mathcal{S}$ analyzes the results and raises a signal s, depending on the correctness of $\mathbf{y}$ and $\mathbf{y}'$. The checkpoint function can therefore be summarized as $s = \mathcal{S}(\mathbf{y}, \mathbf{y}', \mathcal{P})$.

Richter-Brockmann et al. [RBSBG20] proposed an impeccable circuit design for AES. Subsequently, we will use their work as an example to demonstrate how impeccable circuits can be constructed. For encoding, they used the systematic Hamming Code $\mathcal{C}(8, 4, 4)_2$ with the matrix $\mathbf{G}_{4\times 8} = \begin{bmatrix}\mathbf{I}_4 & \mathbf{P}_4\end{bmatrix} \in \mathbb{F}_2^{4\times 8}$, where

$$\mathbf{P}_4 = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \tag{8}$$

Throughout this section, we use $\mathbf{P}$ instead of $\mathbf{P}_4$. Recall that a coding scheme $\mathcal{C}$ can be written as $\mathcal{C}(n, k, d)_q$, where n is the code length, k its cardinality, d the minimum distance, and q the size of the alphabet. For the given code, $n = 8$, $k = 4$, $d = 4$, and $q = 2$ (the code operates on data in the extended binary field $\mathbb{F}_{2^k}$). The choice of the matrix $\mathbf{P}$ has the advantage that it is self-inverse in $\mathbb{F}_2^{4\times 4}$, such that $\mathbf{P} = \mathbf{P}^{-1}$. Therefore , the corresponding parity-check matrix, can be represented as $\mathbf{H}_{8\times 4} = \begin{bmatrix}\mathbf{P} \\ \mathbf{I}\end{bmatrix}$.

Once the code $\mathcal{C}$ is fixed, the redundancy $\mathbf{x}'$ of a binary vector $\mathbf{x} \in \mathbb{F}_{2^k}$ can be calculated as $\mathbf{x}' = \mathbf{x} \cdot \mathbf{P}$. Then the code word $\mathbf{c} = \mathbf{x} \cdot \mathbf{G} = (\mathbf{x} \cdot \mathbf{I}, \mathbf{x} \cdot \mathbf{P}) = (\mathbf{x}, \mathbf{x}')$. So, $\mathcal{P}$ can be defined as $\mathcal{P} : \mathcal{P}(\mathbf{x}) = \mathbf{x} \cdot \mathbf{P}$.

In this paper, we use a slightly modified coding scheme, shown in Equation (9). A matrix like this has also been used in [SMG16], a work that laid the foundation for building secure circuits based on encoded data and that even precedes the original work on impeccable circuits [AMR+20]. The en-/decoding matrix can be represented as:

$$\mathbf{P}' = \begin{bmatrix} \mathbf{P} & \underset{\sim}{\mathbf{0}} & \underset{\sim}{\mathbf{0}} & \cdots & \underset{\sim}{\mathbf{0}} \\ \underset{\sim}{\mathbf{0}} & \mathbf{P} & \underset{\sim}{\mathbf{0}} & \cdots & \underset{\sim}{\mathbf{0}} \\ \underset{\sim}{\mathbf{0}} & \underset{\sim}{\mathbf{0}} & \mathbf{P} & \cdots & \underset{\sim}{\mathbf{0}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \underset{\sim}{\mathbf{0}} & \underset{\sim}{\mathbf{0}} & \underset{\sim}{\mathbf{0}} & \cdots & \mathbf{P} \end{bmatrix}, \tag{9}$$

where $\mathbf{P} = \mathbf{P}_4$ as defined in Equation (8) and $\underset{\sim}{\mathbf{0}}$ is a four-by-four matrix with elements of the value zero. The justification of this choice is given in Section 4.

Now, a given string $\mathbf{c} \in \mathbb{F}_2^{4N}$ can be represented as a set of N words $\mathbf{c}^{(i)}$, such that each $\mathbf{c}^{(i)}$ is encoded with $\mathbf{P}$. Assume, there exist the functions $g(\cdot)$ and $g'(\cdot)$ that apply the function $f(\cdot)$ and the corresponding function $f'(\cdot)$, where:

$$f' = \mathbf{P} \circ f \circ \mathbf{P}^{-1} \tag{10}$$

due to linearity of f, to every $\mathbf{c}^{(i)}$. Therefore, g' can also be defined as:

$$g' = \mathbf{P}' \circ g \circ \mathbf{P}'^{-1} \tag{11}$$

However, this does not hold for non-linear permutations. Subsequently, we demonstrate an AND function for words $\mathbf{c}_1$ and $\mathbf{c}_2$, which are encoded with $\mathbf{P}$. If instead of $\mathbf{P}$, the matrix $\mathbf{P}'$ is used, the approach works analogously by applying the AND gate to each pair of codewords $\mathbf{c}_1^{(i)}, \mathbf{c}_2^{(i)}$. However, before demonstrating our impeccable circuit designs, we formally define an adversary's goal in presence of a coding scheme.

3.3 Fault Attacks on Impeccable Circuits

In this part, we define the success criteria of a fault injection attack on impeccable circuits that use either $\mathbf{P}$ (Equation (8)) or $\mathbf{P}'$ (Equation (9)). First, we revisit the fact that an attacker is able to inject faults into any input, output and intermediate variables. Her goal is to obtain from a given codeword $\mathbf{c} = (\mathbf{x}, \mathbf{x}')$ of a code $\mathcal{C}$ a valid codeword $\mathbf{c}_e \in \mathcal{C}$, such that $\mathbf{c}_e \neq \mathbf{c}$ by introducing an error $\mathbf{e} = (\mathbf{e}_x, \mathbf{e}_{x'})$. If the resulting codeword $\mathbf{c}_e$ is invalid, the modification (the error $\mathbf{e}$) can be detected by the checkpoint or a module with a similar functionality.

3.3.1 Fault Attacks on Linear Impeccable Circuits

For a linear impeccable circuit $\mathcal{F}$ that performs the linear functions $f(\cdot)$ and $f'(\cdot)$, assume, the adversary manipulates output, intermediate and input variables.

Attack on output variables. An impeccable circuit $\mathcal{F}$ produces $\mathbf{c} = (\mathbf{x}, \mathbf{x}')$. Therefore, if the attacker tries to manipulate $\mathbf{c}$ with $\mathbf{e}$, such that $\mathbf{c}_e = \mathbf{c} + \mathbf{e}$, $\mathbf{e}$ has to be a codeword in $\mathcal{C}$, since $\mathbf{c}, \mathbf{c}_e \in \mathcal{C}$. Codewords in $\mathcal{C}$ have a minimum distance d, therefore the Hamming weight of $\mathbf{e}$ must be at least d for the attack to succeed. Therefore, if the attacker tries to manipulate output variables, she has to inject an error $\mathbf{e}$, such that $\mathbf{e} \in \mathcal{C}$ and $HW(\mathbf{e}) \geq d$.

Attack on input variables. An impeccable circuit $\mathcal{F}$ consumes an input $\mathbf{c} = (\mathbf{x}, \mathbf{x}')$ and produces $\mathcal{F}(\mathbf{c}) = (f(\mathbf{x}), f'(\mathbf{x}'))$. The attacker manipulates $\mathbf{c}$ with an error $\mathbf{e} = (\mathbf{e}_x, \mathbf{e}_{x'})$. Therefore,

$$\begin{aligned}\mathbf{c}_e &= (f(\mathbf{x}+\mathbf{e}_x), f'(\mathbf{x}'+\mathbf{e}_{x'}))\\ &= (f(\mathbf{x})+f(\mathbf{e}_x), f'(\mathbf{x}')+f'(\mathbf{e}_{x'}))\\ &= (f(\mathbf{x}), f'(\mathbf{x}')) + (f(\mathbf{e}_x), f'(\mathbf{e}_{x'}))\\ &= \mathcal{F}(\mathbf{c}) + \mathcal{F}(\mathbf{e})\end{aligned}$$

Since $\mathcal{F}(\mathbf{c})$ is a valid codeword and $\mathbf{c}_e$ must be valid as well, the attack is successful, if $\mathcal{F}(\mathbf{e}) \neq \mathbf{0}$ is a valid codeword in $\mathcal{C}$ for a given $\mathbf{e}$.

Attack on intermediate variables. An impeccable circuit $\mathcal{F}$ applies the $f(\cdot)$ and $f'(\cdot)$ functions to its inputs. Moreover f and f' can be decomposed to $f = f_n \circ \cdots \circ f_m \circ \cdots \circ f_2 \circ f_1$ and $f' = f'_n \circ \cdots \circ f'_m \circ \cdots \circ f'_2 \circ f'_1$, such that f_i and f'_i are linear for all $i \in [1, n]$. Assume, that $\mathbf{c} = (\mathbf{x}, \mathbf{x}')$ is the input of $f_m(\cdot)$ and $f'_m(\cdot)$. Let $F = f_n \circ \cdots \circ f_m$ and $F' = f'_n \circ \cdots \circ f'_m$. Therefore, if an error is injected into an intermediate variable $\mathbf{c}$:

$$\begin{aligned}\mathbf{c}_e &= (F(\mathbf{x}+\mathbf{e}_x), F'(\mathbf{x}'+\mathbf{e}_{x'}))\\ &= (F(\mathbf{x})+F(\mathbf{e}_x), F'(\mathbf{x}')+F'(\mathbf{e}_{x'}))\\ &= (F(\mathbf{x}), F'(\mathbf{x}')) + (F(\mathbf{e}_x), F'(\mathbf{e}_{x'}))\\ &= \mathcal{F}_s(\mathbf{c}) + \mathcal{F}_s(\mathbf{e})\end{aligned}$$

where $\mathcal{F}_s \subset \mathcal{F}$. Since $\mathcal{F}_s(\mathbf{c})$ and $\mathbf{c}_e$ are valid codewords, $\mathcal{F}_s(\mathbf{e}) : \mathcal{F}_s(\mathbf{e}) \neq 0$ must be a codeword in $\mathcal{C}$ for a successful attack.

Attack on multiple intermediate variables. Similar to the previous paragraph, we can write $f(\cdot)$ and $f'(\cdot)$ as $f = f_n \circ \cdots \circ f_m \circ \cdots \circ f_2 \circ f_1$ and $f' = f'_n \circ \cdots \circ f'_m \circ \cdots \circ f'_2 \circ f'_1$ and f_i and f'_i are linear for all $i \in [1, n]$. Let $F_1 = f_m \circ \cdots \circ f_2$, $F'_1 = f'_m \circ \cdots \circ f'_2$, $F_2 = f_n \circ \cdots \circ f_{m+1}$, $F'_2 = f'_n \circ \cdots \circ f'_{m+1}$ and let $\mathbf{c} = (\mathbf{x}, \mathbf{x}')$ be the input of F_1 and F'_1. The attacker injects $\mathbf{e}_1 = (\mathbf{e}_{x_1}, \mathbf{e}_{x'_1})$ into $\mathbf{c}$ and $\mathbf{e}_2 = (\mathbf{e}_{x_2}, \mathbf{e}_{x'_2})$ into the input of F_2 and F'_2.

$$\begin{aligned}\mathbf{c}_e &= (F_2(F_1(\mathbf{x}+\mathbf{e}_{x_1})+\mathbf{e}_{x_2}), F'_2(F'_1(\mathbf{x}'+\mathbf{e}_{x'_1})+\mathbf{e}_{x'_2}))\\ &= (F_2(F_1(\mathbf{x})) + F_2(F_1(\mathbf{e}_{x'_1})) + F_2(\mathbf{e}_{x'_2}), F'_2(F'_1(\mathbf{x}')) + F'_2(F'_1(\mathbf{e}_{x'_1})) + F'_2(\mathbf{e}_{x'_2}))\\ &= (F_2(F_1(\mathbf{x})), F'_2(F'_1(\mathbf{x}'))) + (F_2(F_1(\mathbf{e}_{x_1})), F'_2(F'_1(\mathbf{e}_{x'_1}))) + (F_2(\mathbf{e}_{x_2}), F'_2(\mathbf{e}_{x'_2}))\\ &= \mathcal{F}_{s_2}(\mathcal{F}_{s_1}(\mathbf{c})) + \mathcal{F}_{s_2}(\mathcal{F}_{s_1}(\mathbf{e}_1)) + \mathcal{F}_{s_2}(\mathbf{e}_2)\end{aligned}$$

where $\mathcal{F}_{s_1}, \mathcal{F}_{s_2} \subset \mathcal{F}$. Therefore, the codeword $\mathcal{F}_{s_2}(\mathcal{F}_{s_1}(\mathbf{e}_1)) + \mathcal{F}_{s_2}(\mathbf{e}_2) \neq 0$ must be valid in $\mathcal{C}$.

Combination of attacks. An attacker may also combine attacks on different sets of variables. In this case, a recombination of the equations above is required. For example, if for an attack on multiple intermediate variables $F_1 = f_m \circ \cdots \circ f_1$ and $F'_1 = f'_m \circ \cdots \circ f'_1$, the attack combines faults on input and intermediate variables. In general, the attack is successful if $\mathbf{c}_e \in \mathcal{C}$ for a given $\mathbf{e}$.

Encoding with $\mathbf{P'}$. So far, a single impeccable circuit for a linear function processing $\mathbf{x}$ and $\mathbf{x}' = \mathbf{x} \cdot \mathbf{P}$ was considered. However, if $\mathbf{P}'$ is used, each circuit $\mathcal{F}$ consumes, processes and produces multiple codewords, such that $\mathbf{c}_e$ and $\mathbf{e}$ can be represented as a set of codewords. So, the attack criteria are generalized to:

- attack on output variables: $\forall\, \mathbf{e}^{(i)} \in \mathbf{e},\ \mathbf{e}^{(i)} \in \mathcal{C}$,
- attack on input variables: $\forall\, \mathbf{e}^{(i)} \in \mathcal{F}(\mathbf{e}),\ \mathbf{e}^{(i)} \in \mathcal{C}$,
- attack on intermediate variables: $\forall\, \mathbf{e}^{(i)} \in \mathcal{F}_s(\mathbf{e}),\ \mathbf{e}^{(i)} \in \mathcal{C}$,
- attack on multiple intermediate variables: $\forall\, \mathbf{e}^{(i)} \in \mathcal{F}_{s_2}(\mathcal{F}_{s_1}(\mathbf{e}_1)) + \mathcal{F}_{s_2}(\mathbf{e}_2),\ \mathbf{e}^{(i)} \in \mathcal{C}$,
- attack in general: $\forall\, \mathbf{c}_e^{(i)} \in \mathbf{c}_e,\ \mathbf{c}_e^{(i)} \in \mathcal{C}$.

As stated above, a combination of attacks can be represented with a combination of the given equations. It is clear, that if we use $\mathbf{P}$ instead of $\mathbf{P}'$, these criteria are still valid, since $\mathbf{e}$ and $\mathbf{c}_e$ may contain only one element (word).

Active Security of Linear Impeccable Circuits. From our investigations above, we can conclude that the circuit $\mathcal{F}$ is d^{th}-order active secure if for any combination of faults $\mathbf{e}_j$ with $0 < \sum_j HW(\mathbf{e}_j) \leq d$ exists at least one invalid codeword $\mathbf{e}^{(i)} \in \mathbf{e}$ and $\mathbf{e}^{(i)} \notin \mathcal{C}$, such that the fault is detected.

3.3.2 Analyzing the Active Security Order of a Linear Impeccable Circuit

In this example, we demonstrate how the criteria above can be applied to linear impeccable circuits and why their security analysis is important. For this purpose we use a design proposed in [SMG16]. It should be noted, that [SMG16] precedes the general concept of impeccable circuits [AMR+20], while also using the concept of processing redundant encoded data. We consider the Add Constant step in an implementation of the Present [BKL+07] cipher from [SMG16]. An input column vector $\mathbf{r} \in \mathbb{F}_2^6$ is extended with two additional bits, such that $\mathbf{r}' = (\mathbf{r}, 00)$ and encoded with a matrix

$$\mathbf{P}' = \begin{bmatrix} \mathbf{P} & \underset{\sim}{\mathbf{0}} \\ \underset{\sim}{\mathbf{0}} & \mathbf{P} \end{bmatrix}$$

such that $\mathbf{c} = \mathbf{r}' \cdot \mathbf{P}'$. Since the same $\mathbf{P}$ as in [RBSBG20] (described in Section 3.2, recall that $d = 4$) and in this paper is used and the Add Constant step is linear, the circuit should ideally be 3^{rd}-order active secure. The resulting Add Constant matrix is represented as:

$$\mathbf{U}_{L_{check}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix},$$

where $\mathbf{U}_{L_{check}}$is defined in [SMG16].

Remark 3. For any faults in intermediate variables of a vector-matrix multiplication it is easy to see, that a faulted bit does not propagate to more than one output bit. Consider a vector $\mathbf{a} \in \mathbb{R}^n$ and a matrix $\mathbf{B} \in \mathbb{R}^{n \times n}$. Intermediate variables of $\mathbf{a} \cdot \mathbf{B}$ are represented by $a_i b_{i,j}$. These variables are unique and occur only once in the operation.

Since $\mathbf{c}' = \mathbf{c} \cdot \begin{bmatrix} \mathbf{U}_L & \underset{\sim}{\mathbf{0}} \\ \underset{\sim}{\mathbf{0}} & \mathbf{U}_{L_{check}} \end{bmatrix}$, we can formalize an attack on the implementation as:

$$\mathbf{c}_e = \mathbf{c} \cdot \begin{bmatrix} \mathbf{U}_L & \underset{\sim}{\mathbf{0}} \\ \underset{\sim}{\mathbf{0}} & \mathbf{U}_{L_{check}} \end{bmatrix} + \mathbf{e}_1 \cdot \begin{bmatrix} \mathbf{U}_L & \underset{\sim}{\mathbf{0}} \\ \underset{\sim}{\mathbf{0}} & \mathbf{U}_{L_{check}} \end{bmatrix} + \mathbf{e}_2$$

Assume, $\mathbf{e}_1 = (0000000000001000)$. Therefore,

$$\mathbf{e}_1 \cdot \begin{bmatrix} \mathbf{U}_L & \underset{\sim}{\mathbf{0}} \\ \underset{\sim}{\mathbf{0}} & \mathbf{U}_{L_{check}} \end{bmatrix} = (0000000000001101)$$

If $\mathbf{e}_2 = (0000001000000000)$ and

$$\mathbf{e} = \mathbf{e}_1 \cdot \begin{bmatrix} \mathbf{U}_L & \underset{\sim}{\mathbf{0}} \\ \underset{\sim}{\mathbf{0}} & \mathbf{U}_{L_{check}} \end{bmatrix} + \mathbf{e}_2 = (0000001000001101) \equiv (00000000, 00101101) = (\mathbf{e}^{(1)}, \mathbf{e}^{(2)})$$

the attack is successful, since $\mathbf{e}^{(1)}, \mathbf{e}^{(2)} \in \mathcal{C}$ and $HW(\mathbf{e}^{(2)}) > \mathbf{0}$, and therefore, $\mathbf{c}_e^{(1)}, \mathbf{c}_e^{(2)} \in \mathcal{C}$ as $\mathbf{c}_e = \mathbf{c}' + \mathbf{e}$. So, the implementation can be broken with two faults. We can automate the process of finding such errors with an overall Hamming weight smaller than four, which lead to a set of valid codewords. For this purpose we went through all relevant combinations of $\mathbf{e}_1$ and $\mathbf{e}_2$ and checked, whether an output vector $\mathbf{e}$ is non-zero and contains only valid codewords. As a result, we discovered that the implementation can be broken with two faults in five scenarios and three faults in 55 scenarios. In this case, we could disprove 3^{rd}-order active security for the investigated circuit. Moreover, we found that, if the function $f(\cdot)$ is linear, we can analyze the fault propagation for $f(\cdot)$ and $f'(\cdot)$ independent of the processed data.

Example. Assume, the the originally processed data is in $\mathbb{F}_2^{64}$ and $\mathbf{P}' \in \mathbb{F}_2^{64\times 64}$ is used. Therefore, the codeword $\mathbf{c}$ is 128 bit long. This codeword is processed by the impeccable circuit $\mathcal{F}$. Assume the attacker is able to inject up to 10 faults into the input variables of $\mathcal{F}$. Let $\mathbf{e}_{IN}$ be a fault applied to the input $\mathbf{c}$, where $HW(\mathbf{e}_{IN}) \leq 10$. In case of a straightforward simulation, that does not consider that we deal with encoded inputs, one would first have to evaluate all possible values of $\mathbf{e}_{IN}$, which amounts to $\sum_{j=1}^{10} \binom{128}{j}$ possibilities. Further, one would have to simulate all possible values for $\mathbf{e}_{IN}$ with all possible input codewords, so $2^{64} \sum_{j=1}^{10} \binom{128}{j}$ combinations would have to be checked. However, from Section 3.3.1, we know that $\mathbf{e} = \mathcal{F}(\mathbf{e}_{IN})$. This means that the simulation does not need to care about the processed codeword. So now, only $\sum_{j=1}^{10} \binom{128}{j}$ possible error patterns, where $\binom{128}{j}$ corresponds to all errors with a Hamming weight of j, need to be considered. This reduces the verification effort from hardly feasible to a program that can easily be run on a laptop. For each of the possible error patterns, it needs to be checked if $\mathbf{e}$ is either a valid codeword or a set of valid codewords. If this is not the case for any of the possible patterns, the linear impeccable circuit would be 10^{th}-order active secure.

3.3.3 Fault Attacks on Non-Linear Impeccable Circuits

In contrast to linear impeccable circuits, non-linear circuits are more difficult to design and verify for active security. Let $\mathcal{F}$ be an impeccable circuit, such that $f(\cdot)$ and $f'(\cdot)$ are non-linear functions.

Attack on output variables. An attack on the output codeword $\mathbf{c}$ is successful if and only if there exists $\mathbf{e}$ with $HW(\mathbf{e}) > 0$, such that $\mathbf{c}_e = \mathbf{c} + \mathbf{e} \in \mathcal{C} \implies \mathbf{e} \in \mathcal{C}$. This is the same as with linear impeccable circuits.

Attack on input variables. If an error $\mathbf{e} = (\mathbf{e}_x, \mathbf{e}_{x'})$ is injected into the input codeword $\mathbf{c} = (\mathbf{x}, \mathbf{x}')$:

$$\mathbf{c}_e = (f(\mathbf{x} + \mathbf{e}_x), f'(\mathbf{x}' + \mathbf{e}_{x'})) = \mathcal{F}(\mathbf{x} + \mathbf{e}_x, \mathbf{x}' + \mathbf{e}'_x) = \mathcal{F}(\mathbf{c} + \mathbf{e})$$

Since $f(\cdot)$ and $f'(\cdot)$ are non-linear, $\mathcal{F}(\mathbf{c}+\mathbf{e}) \neq \mathcal{F}(\mathbf{c}) + \mathcal{F}(\mathbf{e})$. Therefore, an analysis of errors independent from the processed data is not possible. So, the circuit $\mathcal{F}$ is compromised if there exists an $\mathbf{e}$ with $HW(\mathbf{e}) > 0$, such that $\mathbf{c}_e \in \mathcal{C}$ and $\mathbf{c}_e \neq \mathcal{F}(\mathbf{c})$.

Attack on intermediate variables. If $f(\cdot)$ and $f'(\cdot)$ are composed from other non-linear functions, we can write $f = f_n \circ \cdots \circ f_m \circ \cdots \circ f_1$ and $f' = f'_n \circ \cdots \circ f'_m \circ \cdots \circ f'_1$. Assume the attacker tries to manipulate the inputs of f_m and f'_m. For this purpose, we define $F = f_n \circ \cdots \circ f_m$ and $F' = f'_n \circ \cdots \circ f'_m$ and $\mathbf{x}$, $\mathbf{x}'$ be outputs of f_{m-1} and f'_{m-1} correspondingly, such that $\mathbf{c} = (\mathbf{x}, \mathbf{x}')$. Then,

$$\mathbf{c}_e = (F(\mathbf{x} + \mathbf{e}_x), F'(\mathbf{x}' + \mathbf{e}_{x'})) = \mathcal{F}_s(\mathbf{x} + \mathbf{e}_x, \mathbf{x}' + \mathbf{e}_{x'}) = \mathcal{F}_s(\mathbf{c} + \mathbf{e}),$$

where $\mathcal{F}_s \subset \mathcal{F}$. As above, the criteria for a successful attack given $\mathbf{e}$ with $HW(\mathbf{e}) > 0$ is $\mathbf{c}_e = \mathcal{F}_s(\mathbf{c} + \mathbf{e}) \in \mathcal{C}$ and $\mathbf{c}_e \neq \mathcal{F}_s(\mathbf{c})$.

Attack on multiple intermediate variables. The success condition for an attack on multiple intermediate variables is a generalization of the formula derived in the previous paragraph:

$$\begin{aligned}\mathbf{c}_e &= (F_2(F_1(\mathbf{x} + \mathbf{e}_{x_1}) + \mathbf{e}_{x_2}), F'_2(F'_1(\mathbf{x}' + \mathbf{e}_{x'_1}) + \mathbf{e}_{x'_2})) \\ &= \mathcal{F}_{s_2}(\mathcal{F}_{s_1}(\mathbf{x} + \mathbf{e}_{x_1}, \mathbf{x}' + \mathbf{e}_{x'_1}) + (\mathbf{e}_{x_2}, \mathbf{e}_{x'_2})) \\ &= \mathcal{F}_{s_2}(\mathcal{F}_{s_1}(\mathbf{c} + \mathbf{e}_1) + \mathbf{e}_2),\end{aligned}$$

where $\mathcal{F}_{s_1}, \mathcal{F}_{s_2} \subset \mathcal{F}$. Therefore, a circuit is compromised if a combined manipulation with $\mathbf{e}_1$ and $\mathbf{e}_2$, where $HW(\mathbf{e}_1) + HW(\mathbf{e}_2) > 0$, results in a valid codeword $\mathbf{c}_e \in \mathcal{C}$, such that $\mathbf{c}_e \neq \mathcal{F}_{s_2}(\mathcal{F}_{s_1}(\mathbf{c}))$. The equations for the manipulation of more intermediate variables, as well as the combination of attacks on different variables can be derived from the given description.

Remark 4. As shown above, analysis of non-linear impeccable circuits without the knowledge on the processed data is not possible. Therefore, the verification of non-linear impeccable circuits can not be simplified as discussed in Section 3.3.2.

Encoding with $\mathbf{P}'$. As for linear circuits, an attack's success condition for an encoding with $\mathbf{P}'$ can be written as follows:

- attack on output variables: $\forall\, \mathbf{e}^{(i)} \in \mathbf{e}$, $\mathbf{e}^{(i)} \in \mathcal{C}$,
- attack on other variables: $\forall\, \mathbf{c}^{(i)} \in \mathbf{c}_e$, $\mathbf{c}^{(i)} \in \mathcal{C}$.

Similar to Section 3.3.1, we can now derive a general statement on the active security of non-linear impeccable circuits, even though it does not lead to a simplified verification procedure as with linear impeccable circuits (Section 3.3.2).

Active Security of Non-Linear Impeccable Circuits. An impeccable circuit $\mathcal{F}$ is d^{th}-order active secure if for any combination of errors $\mathbf{e}_j$ with $0 < \sum_j HW(\mathbf{e}_j) \leq d$, the faulty output $\mathbf{c}_e \notin \mathcal{C}$ or $\forall\, \mathbf{c}_e^{(i)} \in \mathbf{c}_e$ exist at least one $\mathbf{c}_e^{(i)} \notin \mathcal{C}$. It is observable that this definition is exactly the same as in Section 3.3.1.

3.4 Fault Attacks on Combined Impeccable Circuits

As stated above, the active security of non-linear impeccable circuits is difficult to verify. Therefore, if a circuit $\mathcal{F}$ consists of a linear $\mathcal{F}_l$ and a non-linear $\mathcal{F}_n$ sub-circuits, the verification complexity cannot be reduced. However, if $\mathcal{F}_n$ is d-SNA (see Section 3.1) for a given $[n, k, d+1]$ linear code, the simplification is possible.

Assume, $\mathcal{F}_n$ is d-SNA. So, for a given $\mathbf{e}$ with $HW(\mathbf{e}) \leq d$ (we assume that $\mathbf{e}$ can be injected into any variables of $\mathcal{F}_n$) holds:

$$\mathbf{c}_e = \mathcal{F}_n(\mathbf{c}, \mathbf{e}) = \mathcal{F}_n(\mathbf{c}) + \mathbf{e}',$$

where $HW(\mathbf{e}') \leq HW(\mathbf{e}) \leq d$ and $\mathbf{e}'$ is the error, which affects the output of $\mathcal{F}_n$ in the same way as $\mathbf{e}$. This is guaranteed by the d-SNA property. Therefore for a combined impeccable circuit $\mathcal{F} = \mathcal{F}_l \circ \mathcal{F}_n$, we can express any fault attacks as:

$$\begin{aligned}\mathbf{c}_e &= \mathcal{F}_l(\mathcal{F}_n(\mathbf{c}, \mathbf{e}_1), \mathbf{e}_2) + \mathbf{e}_3 = \mathcal{F}_l(\mathcal{F}_n(\mathbf{c}) + \mathbf{e}'_1, \mathbf{e}_2) + \mathbf{e}_3 \\ &= \mathcal{F}_l(\mathcal{F}_n(\mathbf{c}) + \mathbf{e}'_1 + \mathbf{e}'_2) + \mathbf{e}_3\end{aligned}$$

where $\mathbf{e}_2$ is mapped to the input of $\mathcal{F}_l$ as $\mathbf{e}'_2$ due to its linearity and $HW(\mathbf{e}'_1) \leq HW(\mathbf{e}_1)$.

$$\begin{aligned}\mathbf{c}_e &= \mathcal{F}_l(\mathcal{F}_n(\mathbf{c})) + \mathcal{F}_l(\mathbf{e}'_1 + \mathbf{e}'_2) + \mathbf{e}_3 \\ &= \mathcal{F}(\mathbf{c}) + \mathcal{F}_l(\mathbf{e}'_1 + \mathbf{e}'_2) + \mathbf{e}_3 = \mathcal{F}(\mathbf{c}) + \mathcal{F}_l(\mathbf{e}'_1, \mathbf{e}_2) \\ &= \mathcal{F}(\mathbf{c}) + \mathbf{e}_{OUT},\end{aligned}$$

where $\mathbf{e}_{OUT} = \mathcal{F}_l(\mathbf{e}'_1 + \mathbf{e}'_2) + \mathbf{e}_3 = \mathcal{F}_l(\mathbf{e}'_1, \mathbf{e}_2) + \mathbf{e}_3$.

Remark 5. Here, we assume that errors $\mathbf{e}_1$ and $\mathbf{e}_2$ can be injected into any variables of $\mathcal{F}_n$ and $\mathcal{F}_l$ correspondingly.

So, $\mathcal{F}$ is compromised if for a set of errors $\{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_N\}$, where $0 < \sum_{i=1}^{N} HW(\mathbf{e}_i)$, $\mathbf{e}_{OUT} \in \mathcal{C}$ (or $\forall \mathbf{e}^{(i)} \in \mathbf{e}_{OUT}$, $\mathbf{e}^{(i)} \in \mathcal{C}$) and $HW(\mathbf{e}_{OUT}) > 0$. Therefore, $\mathcal{F}$ is d^{th}-order active secure if the given condition does not hold for $0 < \sum_{j=1}^{N} HW(\mathbf{e}_j) \leq d$.

3.5 Generalized Impeccable AND Gate

Here, we propose a methodology to construct an impeccable AND gate with the coding scheme from Section 3.2. Our approach avoids look-up tables and is useful for the design of impeccable circuits for KECCAK and various other algorithms.

3.5.1 Construction of f_{AND}

Let $\mathbf{u}$ and $\mathbf{v}$ two 4-bit vectors $\mathbf{u} = (u_0\, u_1\, u_2\, u_3)^T$ and $\mathbf{v} = (v_0\, v_1\, v_2\, v_3)^T$, $\mathbf{u}, \mathbf{v} \in \mathbb{F}_{2^4}$. For simplicity of description, we assume that input vectors are column vectors. All operations on them are done in $\mathbb{F}_{2^4}$. We re-use $\mathbf{P}$ from Equation (8) and $\mathcal{C}(8, 4, 4)_2$. So, the vectors $\mathbf{u}$ and $\mathbf{v}$ are encoded with $\mathbf{G} = \begin{bmatrix}\mathbf{I} \\ \mathbf{P}\end{bmatrix}$. The corresponding codewords $\mathbf{c}_u, \mathbf{c}_v$ can be represented as $\mathbf{c}_u = (\mathbf{u}, \mathbf{u}'), \mathbf{c}_v = (\mathbf{v}, \mathbf{v}')$, where

$$\mathbf{u}' = \mathbf{P} \cdot \mathbf{u} = \begin{pmatrix} u_0 + u_1 + u_2 \\ u_0 + u_1 + u_3 \\ u_0 + u_2 + u_3 \\ u_1 + u_2 + u_3 \end{pmatrix} = \begin{pmatrix} u'_0 \\ u'_1 \\ u'_2 \\ u'_3 \end{pmatrix} \quad \mathbf{v}' = \mathbf{P} \cdot \mathbf{v} = \begin{pmatrix} v_0 + v_1 + v_2 \\ v_0 + v_1 + v_3 \\ v_0 + v_2 + v_3 \\ v_1 + v_2 + v_3 \end{pmatrix} = \begin{pmatrix} v'_0 \\ v'_1 \\ v'_2 \\ v'_3 \end{pmatrix}$$

A straightforward approach for the f_{AND} implementation consists of three steps: decode $\mathbf{c}_u$ and $\mathbf{c}_v$, compute $\mathbf{u} \wedge \mathbf{v}$ and encode the result:

$$\mathbf{r} = \mathbf{G} \cdot ((\mathbf{H} \cdot \mathbf{c}_u) \wedge (\mathbf{H} \cdot \mathbf{c}_v)), \tag{12}$$

where $\mathbf{H} = [\mathbf{P}\ \mathbf{I}]$. It is observable that any error introduced after decoding $\mathbf{c}_u$ and $\mathbf{c}_v$ and before encoding $\mathbf{u} \wedge \mathbf{v}$ will be encoded in $\mathbf{r}$, since $\mathbf{r}$ can be expressed as

$$\mathbf{r} = (\mathbf{u} \wedge \mathbf{v}, \mathbf{P} \cdot (\mathbf{u} \wedge \mathbf{v})) = (\mathbf{u} \wedge \mathbf{v}, \mathbf{P} \cdot ((\mathbf{P} \cdot \mathbf{u}') \wedge (\mathbf{P} \cdot \mathbf{v}')))$$

Obviously the processing of decoded data is a vulnerability that must be avoided at all costs. Therefore, we define a function f_{AND} that performs AND operation on encoded data without the decoding:

$$\mathbf{r} = (\mathbf{u} \wedge \mathbf{v}, f_{AND}(\mathbf{u}', \mathbf{v}'))$$

Assume, $\exists\, f_{AND} : (\mathbf{u}' \wedge \mathbf{v}') \mapsto \mathbf{P} \cdot (\mathbf{u} \wedge \mathbf{v})$. We can see that

$$\begin{aligned}\mathbf{u}' \wedge \mathbf{v}' &= \begin{pmatrix} u_0+u_1+u_2 \\ u_0+u_1+u_3 \\ u_0+u_2+u_3 \\ u_1+u_2+u_3 \end{pmatrix} \wedge \begin{pmatrix} v_0+v_1+v_2 \\ v_0+v_1+v_3 \\ v_0+v_2+v_3 \\ v_1+v_2+v_3 \end{pmatrix} = \begin{pmatrix} (u_0+u_1+u_2)(v_0+v_1+v_2) \\ (u_0+u_1+u_3)(v_0+v_1+v_3) \\ (u_0+u_2+u_3)(v_0+v_2+v_3) \\ (u_1+u_2+u_3)(v_1+v_2+v_3) \end{pmatrix} \\ &= \begin{pmatrix} u_0v_0+u_0v_1+u_0v_2+u_1v_0+u_1v_1+u_1v_2+u_2v_0+u_2v_1+u_2v_2 \\ u_0v_0+u_0v_1+u_0v_3+u_1v_0+u_1v_1+u_1v_3+u_3v_0+u_3v_1+u_3v_3 \\ u_0v_0+u_0v_2+u_0v_3+u_2v_0+u_2v_2+u_2v_3+u_3v_0+u_3v_2+u_3v_3 \\ u_1v_1+u_1v_2+u_1v_3+u_2v_1+u_2v_2+u_2v_3+u_3v_1+u_3v_2+u_3v_3 \end{pmatrix}\end{aligned} \tag{13}$$

has to be mapped to

$$\mathbf{P} \cdot (\mathbf{u} \wedge \mathbf{v}) = \mathbf{P} \cdot \begin{pmatrix} u_0v_0 \\ u_1v_1 \\ u_2v_2 \\ u_3v_3 \end{pmatrix} = \begin{pmatrix} u_0v_0+u_1v_1+u_2v_2 \\ u_0v_0+u_1v_1+u_3v_3 \\ u_0v_0+u_2v_2+u_3v_3 \\ u_1v_1+u_2v_2+u_3v_3 \end{pmatrix} \tag{14}$$

Therefore, we define a correction function $\mathcal{Y}(\mathbf{u}', \mathbf{v}')$, such that

$$f_{AND}(\mathbf{u}' \wedge \mathbf{v}') = \mathbf{u}' \wedge \mathbf{v}' + \mathcal{Y}(\mathbf{u}', \mathbf{v}') = \mathbf{u}' \wedge \mathbf{v}' + \mathbf{y} \tag{15}$$

From Equation (15) follows

$$\mathbf{y} = \mathbf{P} \cdot (\mathbf{u} \wedge \mathbf{v}) + \mathbf{u}' \wedge \mathbf{v}' \tag{16}$$

Therefore,

$$\mathbf{u}' \wedge \mathbf{v}' + \mathbf{y} = \mathbf{u}' \wedge \mathbf{v}' + \underbrace{\begin{pmatrix} u_0v_1+u_0v_2+u_1v_0+u_1v_2+u_2v_0+u_2v_1 \\ u_0v_1+u_0v_3+u_1v_0+u_1v_3+u_3v_0+u_3v_1 \\ u_0v_2+u_0v_3+u_2v_0+u_2v_3+u_3v_0+u_3v_2 \\ u_1v_2+u_1v_3+u_2v_1+u_2v_3+u_3v_1+u_3v_2 \end{pmatrix}}_{\mathbf{y}} = \mathbf{P} \cdot (\mathbf{u} \wedge \mathbf{v})$$

This shows that f_{AND} can be constructed from a normal AND gate and a correction vector $\mathbf{y} = \mathcal{Y}(\mathbf{u}', \mathbf{v}')$. The latter can be written as

$$\mathbf{y} = \begin{pmatrix} u_0(v_1+v_2)+u_1(v_0+v_2)+u_2(v_0+v_1) \\ u_0(v_1+v_3)+u_1(v_0+v_3)+u_3(v_0+v_1) \\ u_0(v_2+v_3)+u_2(v_0+v_3)+u_3(v_0+v_2) \\ u_1(v_2+v_3)+u_2(v_1+v_3)+u_3(v_1+v_2) \end{pmatrix}$$

We see that, at least for now, we only shifted the problem of processing decoded data to $\mathbf{y}$.

However, we can also construct $\mathbf{y}$ using only $\mathbf{u}'$ and $\mathbf{v}'$. For this, we first extend $\mathbf{y}$:

$$\begin{aligned}\mathbf{y} &= \begin{pmatrix} u_0v_1 + u_0v_2 + u_1v_0 + u_1v_2 + u_2v_0 + u_2v_1 + u_0v_0 \\ u_0v_1 + u_0v_3 + u_1v_0 + u_1v_3 + u_3v_0 + u_3v_1 + u_0v_0 \\ u_0v_2 + u_0v_3 + u_2v_0 + u_2v_3 + u_3v_0 + u_3v_2 + u_0v_0 \\ u_1v_2 + u_1v_3 + u_2v_1 + u_2v_3 + u_3v_1 + u_3v_2 + u_1v_1 \end{pmatrix} \\ &+ \begin{pmatrix} u_0v_0 + u_0v_0 + u_1v_1 + u_1v_1 + u_2v_2 + u_2v_2 \\ u_0v_0 + u_0v_0 + u_1v_1 + u_1v_1 + u_3v_3 + u_3v_3 \\ u_0v_0 + u_0v_0 + u_2v_2 + u_2v_2 + u_3v_3 + u_3v_3 \\ u_1v_1 + u_1v_1 + u_2v_2 + u_2v_2 + u_3v_3 + u_3v_3 \end{pmatrix} \\ &= \begin{pmatrix} (u_0+u_1)(v_0+v_1) + (u_0+u_2)(v_0+v_2) + (u_1+u_2)(v_1+v_2) \\ (u_0+u_1)(v_0+v_1) + (u_0+u_3)(v_0+v_3) + (u_1+u_3)(v_1+v_3) \\ (u_0+u_2)(v_0+v_2) + (u_0+u_3)(v_0+v_3) + (u_2+u_3)(v_2+v_3) \\ (u_1+u_2)(v_1+v_2) + (u_1+u_3)(v_1+v_3) + (u_2+u_3)(v_2+v_3) \end{pmatrix}\end{aligned}$$

The decomposition of $\mathbf{y}$ gives:

$$\mathbf{D} = \begin{bmatrix} (u_0+u_1)(v_0+v_1) & (u_0+u_2)(v_0+v_2) & (u_1+u_2)(v_1+v_2) \\ (u_0+u_1)(v_0+v_1) & (u_0+u_3)(v_0+v_3) & (u_1+u_3)(v_1+v_3) \\ (u_0+u_2)(v_0+v_2) & (u_0+u_3)(v_0+v_3) & (u_2+u_3)(v_2+v_3) \\ (u_1+u_2)(v_1+v_2) & (u_1+u_3)(v_1+v_3) & (u_2+u_3)(v_2+v_3) \end{bmatrix} \tag{17}$$

$$y_i = \sum_j D_{i,j} \quad \forall y_i \in \mathbf{y} \tag{18}$$

The matrix $\mathbf{D}$ can be decomposed further into two other matrices $\mathbf{U}$ and $\mathbf{V}$:

$$\mathbf{U} = \begin{bmatrix} u_0+u_1 & u_0+u_2 & u_1+u_2 \\ u_0+u_1 & u_0+u_3 & u_1+u_3 \\ u_0+u_2 & u_0+u_3 & u_2+u_3 \\ u_1+u_2 & u_1+u_3 & u_2+u_3 \end{bmatrix} \quad \mathbf{V} = \begin{bmatrix} v_0+v_1 & v_0+v_2 & v_1+v_2 \\ v_0+v_1 & v_0+v_3 & v_1+v_3 \\ v_0+v_2 & v_0+v_3 & v_2+v_3 \\ v_1+v_2 & v_1+v_3 & v_2+v_3 \end{bmatrix}$$

such that:

$$\mathbf{D} = \mathbf{U} \wedge \mathbf{V} : D_{i,j} = U_{i,j} \cdot V_{i,j} \tag{19}$$

Further, $\mathbf{U}$ and $\mathbf{V}$ can be obtained from $\mathbf{u}'$ and $\mathbf{v}'$ directly:

$$\mathbf{U} = \begin{bmatrix} u'_2+u'_3 & u'_1+u'_3 & u'_1+u'_2 \\ u'_2+u'_3 & u'_0+u'_3 & u'_0+u'_2 \\ u'_1+u'_3 & u'_0+u'_3 & u'_0+u'_1 \\ u'_1+u'_2 & u'_0+u'_2 & u'_0+u'_1 \end{bmatrix} \quad \mathbf{V} = \begin{bmatrix} v'_2+v'_3 & v'_1+v'_3 & v'_1+v'_2 \\ v'_2+v'_3 & v'_0+v'_3 & v'_0+v'_2 \\ v'_1+v'_3 & v'_0+v'_3 & v'_0+v'_1 \\ v'_1+v'_2 & v'_0+v'_2 & v'_0+v'_1 \end{bmatrix} \tag{20}$$

such that:

$$U_{i,j} = \sum_{\substack{k=0 \\ k\neq i \\ k\neq 2-j}}^{3} u'_k \quad V_{i,j} = \sum_{\substack{k=0 \\ k\neq i \\ k\neq 2-j}}^{3} v'_k \tag{21}$$

So, Equation (21), Equation (19) and Equation (18) allow the computation of $\mathbf{y}$ from $\mathbf{u}'$ and $\mathbf{v}'$ and, therefore, evaluate an AND operation without decoding data, such that

$$\mathbf{c}_{u\wedge v} = (\mathbf{u} \wedge \mathbf{v}, f_{AND}(\mathbf{u}', \mathbf{v}')), \tag{22}$$

where $\mathbf{c}_{u\wedge v} = \mathbf{r}$ from Equation (12). This approach works for any square matrix $\mathbf{P}_k$ with zeros on the off-diagonal and ones in other positions, where $k = 2\mathbf{N}$, and which is a part of a generator matrix. Moreover, from the AND gate, we can easily construct other non-linear gates such as an OR gate, since $\mathbf{u} \vee \mathbf{v} = (\mathbf{u} + \mathbf{v}) + \mathbf{u} \wedge \mathbf{v}$.

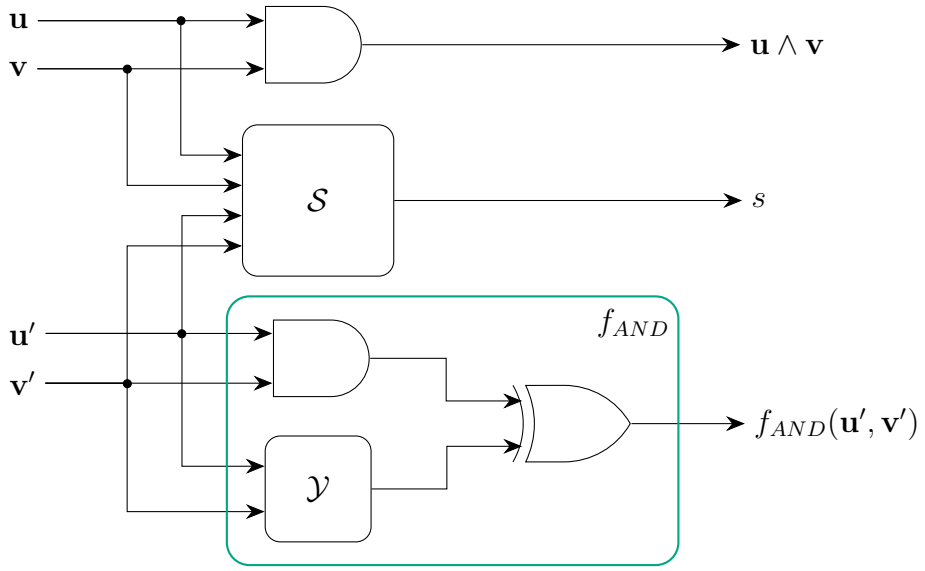


Figure 4: Impeccable AND gate. Operands: $\mathbf{c}_u = (\mathbf{u}|\mathbf{u}')$ and $\mathbf{c}_v = (\mathbf{v}|\mathbf{v}')$. Result: $\mathbf{r} = (\mathbf{u} \wedge \mathbf{v}|\mathbf{P} \cdot (\mathbf{u} \wedge \mathbf{v}))$. The checkpoint compares decoded and encoded data to ensure the integrity of the input. If the integrity is violated, the error signal s is raised to abort the operation.

3.5.2 Active Security of f_{AND}

We established that we can construct an impeccable AND gate by applying f_{AND} to the encoded data independently, i.e. without introducing new dependencies between the decoded and encoded data processed by $\mathcal{F}$. However, this does not mean that the code's minimum distance d is still the lower bound for bit faults an attacker needs to corrupt the AND gate. In fact, it can be seen that a single bit error in $\mathbf{u}'$ or $\mathbf{v}'$ flips up to three output bits of f_{AND} (Equation (20)). Accordingly, the adversary would have to flip only one additional bit in either $\mathbf{u}$, $\mathbf{v}$ or $\mathbf{u} \wedge \mathbf{v}$ to successfully manipulate the circuit with a total of two faults. Therefore, using the concept from [DN20], the circuit is only 1^{st}-order active-secure.

However, if we insert a checkpoint before the f_{AND} function, it guarantees the integrity of $\mathbf{c}_u = (\mathbf{u}, \mathbf{u}')$ and $\mathbf{c}_v = (\mathbf{v}, \mathbf{v}')$. The overall structure of our impeccable AND gate is shown in Figure 4. For the original AND operation, a single bit-flip in input variables does not affect more than one output bit. The security of the checkpoint itself is analyzed in Section 3.5.3. Therefore, we assume that inputs are secured against fault injection up to order d and analyze the remaining part of the circuit, which is described by Equations (18), (19) and (21).

If the attacker injects a single bit error into $\mathbf{y}$, $\exists!\, y_i \in \mathbf{y} : y_i \leftarrow y_i + 1$ that implies $\exists!\, r_i \in \mathbf{r} : r_i \leftarrow r_i + 1$ (Equation (15)). So, a single fault in $\mathbf{y}$ corrupts only one output bit in $\mathbf{r}$.

If the attacker injects a single bit error into $\mathbf{D}$, such that $\exists!\, D_{i,j} \leftarrow D_{i,j} + 1$ (Equation (19)), then $\exists!\, y_i \in \mathbf{y} : y_i \leftarrow y_i + 1 = 1 + \sum_j D_{i,j}$ (Equation (18)), which does not affect more than one bit of $\mathbf{r}$.

If the attacker injects a single fault into $\mathbf{U}$, such that $\exists!\, U_{i,j} \leftarrow U_{i,j} + 1$, then $\exists!\, D_{i,j} \in \mathbf{D} : D_{i,j} = (U_{i,j} + 1) \wedge V_{i,j}$ that implies $\exists^{\leq 1} D_{i,j} \leftarrow D_{i,j} + 1$ (Equation (19), Equation (18)), which does not affect more than one bit of $\mathbf{r}$. A single bit-flip of $\mathbf{V}$ propagates in the same way and also does not flip more than one bit of $\mathbf{r}$. Since the distance of $\mathcal{C}$ is 4 and no error

alters more than one output bit, the function f_{AND} is 3^{rd}-order active secure.

Remark 6. This proof can be extended for a given Hamming Code and a square matrix $\mathbf{P}_k$ with zeros on the off-diagonal and ones in other positions, where $k = 2m$, with $m \in \mathbb{Z}$.

3.5.3 Active Security of the Checkpoint

As shown above, the AND operation is 3^{rd}-order active secure, if the checkpoint is able to detect up to three input faults. Therefore, a combination of AND and f_{AND} is 3^{rd}-order active secure, if for any up to three faults, the circuit produces a non-valid codeword or an error signal.

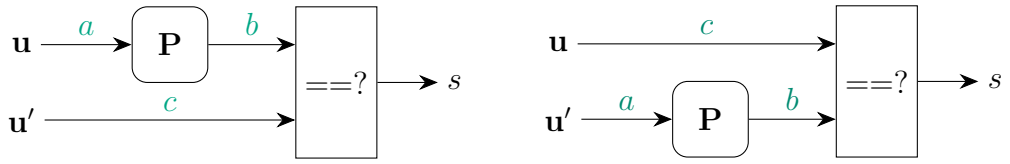
The checkpoint $\mathcal{S}$ can be implemented in two ways: it either computes $\mathbf{u}' \times \mathbf{P}$ and compares it with $\mathbf{u}$ (Checkpoint 1), or $\mathbf{u} \times \mathbf{P}$ and compares it with $\mathbf{u}'$ (Checkpoint 2). In this section we analyze the order of active security for both methods. For this analysis, we first describe how an error propagates through $\mathbf{P}$. Since this corresponds to a generic vector-matrix multiplication, it is easy to see that $\mathbf{e}_{OUT} = \mathbf{e}_{IN} \cdot \mathbf{P}$ (Section 3.3.1), therefore, for any input error $\mathbf{e}_{IN}$ with $HW(\mathbf{e}_{IN}) \leq d$ holds:

$$HW(\mathbf{e}_{OUT}) = \begin{cases} 0, \text{ if } HW(\mathbf{e}_{IN}) = 0 \\ 1, \text{ if } HW(\mathbf{e}_{IN}) = 3 \\ 2, \text{ if } HW(\mathbf{e}_{IN}) = 2 \\ 3, \text{ if } HW(\mathbf{e}_{IN}) = 1 \\ 4, \text{ if } HW(\mathbf{e}_{IN}) = 4 \end{cases}$$

Hence, faults in intermediate variables are equivalent to faults in output variables in this case. For faults in output variables, it is easy to see that the observations from Section 3.3.1 and Section 3.3.3 hold. We also emphasize that the comparator outputs 0, if inputs b and c are equal, 1 otherwise. Therefore:

$$s = \begin{cases} 0, \text{ if } val(b) \oplus val(c) = 0 \iff HW(val(b) \oplus val(c)) = 0 \\ 1 \text{ otherwise} \end{cases}$$

Since all calculations are done in the extended binary field, $\oplus$ is replaced by $+$.



(a) Checkpoint 1. **(b)** Checkpoint 2.

Figure 5: The internal structure of the checkpoint. $\mathbf{u}$ and $\mathbf{u}'$ are parts of the codeword $\mathbf{c} = (\mathbf{u}, \mathbf{u}')$. The signal $\mathbf{s}$ is the output that is equal to 0, if comparator inputs are the same, 1 otherwise. Wires (communication channels) are denoted by a, b and c.

Checkpoint 1. A representation of Checkpoint 1 is shown in Figure 5a. Note that the checkpoint is implemented before the f_{AND} function. Therefore, an error injected into $\mathbf{u}$ or $\mathbf{u}'$ propagates either to the AND or the f_{AND} gate via a or c. Therefore, the attacker is required to manipulate both $\mathbf{u}$ and $\mathbf{u}'$. So, if a single bit in $\mathbf{u}$ and $\mathbf{u}'$ is flipped, the corresponding values $\mathbf{u}+\mathbf{e}_u$ and $\mathbf{u}'+\mathbf{e}_{u'}$ are inputs of the AND and f_{AND} gate respectively. For the AND gate, a single fault on the input does not affect more than one output variable, for the f_{AND} gate less than four bits are corrupted (Equations (18), (19) and (21)). In

the worst case, the attacker flips four output bits in total and is able to obtain another valid codeword. However, this is not the case, if the checkpoint is implemented. A single fault injected into $\mathbf{u}$ propagates to $\mathbf{P}$ and flips exactly three of its output bits. Since only one bit in $\mathbf{u}'$ is manipulated, $\mathbf{u} \cdot \mathbf{P}$ and $\mathbf{u}'$ cannot be equal, since

$$(\mathbf{u} \cdot \mathbf{P} + \mathbf{e}_u \cdot \mathbf{P}) + \mathbf{u}' + \mathbf{e}_{u'} = \mathbf{u}' + \mathbf{e}_u \cdot \mathbf{P} + \mathbf{u}' + \mathbf{e}_{u'} = \mathbf{e}_u \cdot \mathbf{P} + \mathbf{e}_{u'}$$

and $HW(\mathbf{e}_u \cdot \mathbf{P}) = 3$, while $HW(\mathbf{e}_{u'}) = 1$, which implies

$$HW(\mathbf{e}_u \cdot \mathbf{P} + \mathbf{e}_{u'}) = HW(\mathbf{e}_u \cdot \mathbf{P}) + HW(\mathbf{e}_{u'}) - 2HW(\mathbf{e}_u \cdot \mathbf{P} \wedge \mathbf{e}_{u'}) \geq 2$$

So, the attacker needs at least two additional faults either for $\mathbf{u} \times \mathbf{P}$ or for $\mathbf{u}'$, and therefore, four faults in total. A similar analysis can be conducted for other scenarios. If the attacker manipulates two bits of $\mathbf{u}$ and exactly one bit of $\mathbf{u}'$, then $HW(\mathbf{e}_u \cdot \mathbf{P}) = 2$, while $HW(\mathbf{e}_{u'}) = 1$. Therefore:

$$HW(\mathbf{e}_u \cdot \mathbf{P} + \mathbf{e}_{u'}) = HW(\mathbf{e}_u \cdot \mathbf{P}) + HW(\mathbf{e}_{u'}) - 2HW(\mathbf{e}_u \cdot \mathbf{P} \wedge \mathbf{e}_{u'}) \geq 1$$

and the attacker has to inject at least one additional fault into either $\mathbf{u} \cdot \mathbf{P}$ or $\mathbf{u}'$ to bypass the security check. In general, four faults are necessary for the given scenario. If she flips one bit in $\mathbf{u}$ and two bits in $\mathbf{u}'$, then $HW(\mathbf{e}_u \cdot \mathbf{P}) = 1$ and $HW(\mathbf{e}_{u'}) = 2$ implies

$$HW(\mathbf{e}_u \cdot \mathbf{P} + \mathbf{e}_{u'}) = HW(\mathbf{e}_u \cdot \mathbf{P}) + HW(\mathbf{e}_{u'}) - 2HW(\mathbf{e}_u \cdot \mathbf{P} \wedge \mathbf{e}_{u'}) \geq 1$$

and forces the attacker to flip one more bit in one of two comparator inputs, which corresponds to four faults in total. Therefore, this checkpoint implementation in combination with an AND gate is 3^{rd}-order active-secure.

Checkpoint 2. An implementation of Checkpoint 2 is depicted in Figure 5b. The attacker still has to manipulate both data paths for a successful attack. Security analysis for this case is similar to the analysis above:

$$\mathbf{u} + \mathbf{e}_u + (\mathbf{u}' + \mathbf{e}_{u'}) \cdot \mathbf{P} \stackrel{!}{=} 0$$
$$\mathbf{e}_u + \mathbf{e}'_u \cdot \mathbf{P} \stackrel{!}{=} 0 \implies HW(\mathbf{e}_u + \mathbf{e}'_u \cdot \mathbf{P}) \stackrel{!}{=} 0$$

If the attacker flips a single variable in $\mathbf{u}$, this fault propagates to the AND gate and flips at most one of its output bits. However, if she flips one bit in $\mathbf{u}'$, it affects at most three bits on the output of f_{AND} and exactly three bits in $\mathbf{u}' \cdot \mathbf{P}$. The latter means

$$HW(\mathbf{e}_u + \mathbf{e}'_u \cdot \mathbf{P}) = HW(\mathbf{e}_u) + HW(\mathbf{e}'_u \cdot \mathbf{P}) + HW(\mathbf{e}'_u \cdot \mathbf{P} \wedge \mathbf{e}_u) \geq 2$$

Therefore, the attacker must inject two more faults into $\mathbf{u}$ or $\mathbf{u}' \cdot \mathbf{P}$, which sums up to four faults in total. In the next scenario, the attacker flips two bits in $\mathbf{u}$ and one bit in $\mathbf{u}'$. This pattern does also not lead to a successful attack, since:

$$HW(\mathbf{e}_u + \mathbf{e}'_u \cdot \mathbf{P}) = HW(\mathbf{e}_u) + HW(\mathbf{e}'_u \cdot \mathbf{P}) + HW(\mathbf{e}'_u \cdot \mathbf{P} \wedge \mathbf{e}_u) \geq 1$$

So, the attacker has to inject one more error into one of comparator inputs (four in total). In the last scenario, the attacker manipulates one variable in $\mathbf{u}$ and two variables in $\mathbf{u}'$. In this case

$$HW(\mathbf{e}_u + \mathbf{e}'_u \cdot \mathbf{P}) = HW(\mathbf{e}_u) + HW(\mathbf{e}'_u \cdot \mathbf{P}) + HW(\mathbf{e}'_u \cdot \mathbf{P} \wedge \mathbf{e}_u) \geq 1$$

this forces the attacker to flip one more bit in b or c. In total, this scenario requires four faults as well. Therefore, this implementation of the checkpoint is also 3^{rd} order active secure.

Remark 7. Note that the adversary is able to manipulate values in b and c channels as well. Therefore, if the same fault is injected into them, the comparator is not capable to detect this error. So, comparator can be "switched off" with two faults. However, the channel c is directly connected to either the AND or f_{AND} gate and the corresponding output will contain an error, while the other data path is error free, since channels a and b are isolated via $\mathbf{P}$. Therefore, an error can be detected in the subsequent operation.

It should be noted that the checkpoint's output signal s is out-of-scope for the analysis, as it is trivial to protect with multi rails and a sparse encoding.

3.5.4 Arbitrary-order Non-linear Impeccable Circuits

As shown above, for up to three faults on input and intermediate variables, the AND gate either produces an output with at most three errors or an error signal. Therefore, the proposed structure is 3-SNA. From [DN20], we know that a combination of 3-SNA circuits is 3-NA and also 3^{rd}-order active secure. Therefore, a combination of proposed AND gates is 3^{rd}-order active secure. However, this only applies if each AND gate has its own checkpoint. Since this makes our impeccable AND gate rather expensive in terms of hardware consumption, it is expected that implementing algorithms with a high algebraic degree such as AES is expensive with relation to resource utilization. For simple S-boxes with only one AND operation, such as the χ permutation in Keccak, we will show in Section 4 and Section 5 that the overhead is feasible.

4 Encoded Keccak Round Function

In this section, our design of an impeccable Keccak is presented and its active security is proven. For this, we re-use the concepts introduced in Section 3. In particular, we provide reasoning for the chosen coding scheme that we already introduced in Section 3.2 (Equation (9)). Further, we re-use our impeccable AND gate (Section 3.5) and our investigation of linear and non-linear impeccable circuits (Sections 3.3.1 and 3.3.3) to construct impeccable circuits for the five steps within a Keccak round and to prove their order of active security.

Encoding the Keccak state For a general matrix $\mathbf{P}$ with a pattern as in Equation (8) to have an inverse $\mathbf{P}^{-1}$ and $\mathbf{P}^{-1} = \mathbf{P}$, it must hold that $k \in 2\mathbb{N}$. If the complete Keccak state were to be encoded with a 1600×1600 matrix $\mathbf{P}_{1600}$, the matrix alone would require 312.5kB of storage. Further, the construction of θ', ρ', π' and χ' would raise significant implementation issues due to complex inter-bit dependencies. The encoding of sheets or planes would still imply huge dimensions for $\mathbf{P}$ and have similar difficulties regarding the realization of some steps within the Keccak permutation, which are then non-linear with respect to the encoded sub-structure. As slices and rows span an uneven amount of bits, they would have to be padded, which introduces an overhead. An encoding of lanes seems more promising, as it allows an efficient implementation of all permutations, with χ being the only non-linear permutation. If lanes were encoded directly with $\mathbf{P}_{64}$, 12,5kB for the matrix storage would be required. We can reduce this by dividing each lane into multiple sublanes of length 4. This allows us to use a matrix $\mathbf{P}'$ as was previously described in Section 3.2. Since this matrix is sparse, vector-matrix multiplication in hardware is promising due to low number of interconnected variables. For the remainder of this paper, we describe an impeccable Keccak implementation that uses this encoding. For simplicity we also denote $\mathbf{P}'$ as $\mathbf{P}$ throughout this and the next sections. We start by describing ρ, as its results can be reused subsequently.

4.1 ρ permutation

In vector form, Equation (3) can be written as:

$$\rho(\mathbf{a}_{x,y}) = \mathbf{a}_{x,y} \cdot \mathbf{R}^{(t)} \tag{23}$$

where $\mathbf{a}_{x,y} \in \mathbb{F}_{2^{64}}$ is a 64-bit lane of the KECCAK state and $\mathbf{R}^{(t)}$ is the rotation matrix in $\mathbb{F}_2^{64\times 64}$. In the following, we omit (x, y) indices for simplicity. According to Figure 3, we have ρ and its redundant block computing on encoded data ρ', therefore ρ' processes $\mathbf{a}' = \mathbf{a} \cdot \mathbf{P}$. With Equation (10), $\rho' = \mathcal{P} \circ \rho \circ \mathcal{P}^{-1}$. We can merge this into a single vector-matrix multiplication:

$$\rho'(\mathbf{a}') = \mathbf{a}' \cdot \mathbf{T}^{(t)} = \mathbf{a}' \cdot \mathbf{P}^{-1} \cdot \mathbf{R}^{(t)} \cdot \mathbf{P} = \mathbf{a}' \cdot \mathbf{P} \cdot \mathbf{R}^{(t)} \cdot \mathbf{P} \tag{24}$$

Active Security of $\mathcal{F}_\rho$. Since ρ and ρ' are applied lane-wise with different rotation coefficients, we analyze their security with a lane granularity. Due to linearity of $\mathcal{F}_\rho$, we use the strategy from Section 3.3.1 to investigate its attack surface. According to Equation (23) and Equation (24), both ρ and ρ' can be expressed as vector-matrix multiplications. An attack on the output variables is straightforward and requires the manipulation of at least four variables (bits). The attack on intermediate results of vector-matrix multiplication can be expressed as an attack on its output variables (Section 3.3.2) and, therefore, is straightforward as well.

It is unclear, what effect faulting input variables (in combination with other variables as well) has, therefore we analyze it in more detail. Outputs of ρ and ρ' can be written as $\left(\mathbf{a} \cdot \mathbf{R}^{(t)}, \mathbf{a}' \cdot \mathbf{T}^{(t)}\right)$. Based on our analysis in Section 3.3.1, we can express fault attacks on input, intermediate, and output values in terms of the resulting error $\mathbf{e} = (\mathbf{e}_{a_1} \cdot \mathbf{R}^{(t)}, \mathbf{e}_{a'_1} \cdot \mathbf{T}^{(t)}) + (\mathbf{e}_{a_2}, \mathbf{e}_{a'_2})$. The error vector $\mathbf{e}_1 = (\mathbf{e}_{a_1}, \mathbf{e}_{a'_1})$ covers errors within input variables. Errors within intermediate and output variables are expressed with $\mathbf{e}_2 = (\mathbf{e}_{a_2}, \mathbf{e}_{a'_2})$. For a successful attack, $\mathbf{e}$ has to represent a set of valid codewords in $\mathcal{C}$ and $HW(\mathbf{e}) > 0$. This condition can be rewritten in form of the following equation:

$$\mathbf{e} = \mathbf{e}_1 \cdot \begin{bmatrix} \mathbf{R}^{(t)} & \underset{\sim}{\mathbf{0}} \\ \underset{\sim}{\mathbf{0}} & \mathbf{T}^{(t)} \end{bmatrix} + \mathbf{e}_2 : \forall \mathbf{e}^{(i)} \in \mathbf{e} \quad \mathbf{e}^{(i)} \in \mathcal{C} \tag{25}$$

If, for Equation (25), there does not exist a pair of vectors $\mathbf{e}_1$, $\mathbf{e}_2$, where $HW(\mathbf{e}_1) + HW(\mathbf{e}_2) < 4$, that can be exploited by the attacker, $\mathcal{F}_\rho$ is 3^{rd}-order active secure (Section 3.3.1). For this purpose, we analyze all relevant combinations of $\mathbf{e}_1$ and $\mathbf{e}_2$ and verify the breakdown condition for each resulting $\mathbf{e}$. As described in Section 3.3.2, our approach makes the verification computationally feasible as it vastly limits the amount of combinations that need to be evaluated from all possible values of $\mathbf{e}_1$ and $\mathbf{e}_2$. In this case, $\binom{128}{1} + 4\binom{128}{2} + 2\binom{128}{3} = 716,032$ different combinations have to be evaluated for each of the 24 possible values of t. Fortunately, for no combination with $0 < \sum_j HW(\mathbf{e}_j) < 4$, Equation (25) holds. Therefore, $\mathcal{F}_\rho$ is 3^{rd}-order active secure.

4.2 θ permutation

For a lane $\mathbf{a}_{x,y}$, the θ permutation (Equation (2)) can be rewritten in vector form:

$$\theta(\mathbf{a}_{x,y}) = \mathbf{a}_{x,y} \oplus \bigoplus_{y=0}^{4} \mathbf{a}_{x-1,y} \oplus \left(\bigoplus_{y=0}^{4} \mathbf{a}_{x+1,y} \right) \cdot \mathbf{R}^{(1)} \tag{26}$$

For an encoded lane $\mathbf{a}'_{x,y}$, only the matrix multiplication with $\mathbf{R}^{(1)}$ must be adjusted similarly to Equation (24). The exclusive or operations do not require any changes.

Therefore, the equation for $\mathbf{a}'$ can be written as:

$$\theta'(\mathbf{a}'_{x,y}) = \mathbf{a}'_{x,y} \oplus \bigoplus_{y=0}^{4} \mathbf{a}'_{x-1,y} \oplus \left(\bigoplus_{y=0}^{4} \mathbf{a}'_{x+1,y} \right) \cdot \mathbf{T}^{(1)} \tag{27}$$

Active Security of $\mathcal{F}_\theta$. Both θ and θ' consist of addition in $\mathbb{F}_2^{64}$ and a vector-matrix multiplication. A single bit fault injected during addition does not corrupt more than one output variable. Moreover, vector-matrix multiplication with $\mathbf{R}^{(1)}$ and $\mathbf{R}^{(1)}$ was already discussed in Section 4.1. Therefore, $\mathcal{F}_\theta$ has the same order of active security as $\mathcal{F}_\rho$.

4.3 π permutation

The π permutation does not require any modifications, since it only shuffles complete lanes. We can express this with a simple change of (x, y) coordinates:

$$\pi'(\mathbf{a}'_{x,y}) = \mathbf{a}'_{x',y'} : (x, y) = (y', 2x' + 3y') \tag{28}$$

Active Security of $\mathcal{F}_\pi$. Since π and π' are identical and require only rewiring according to the lane coordinate, they do not accumulate any errors. Therefore, $\mathcal{F}_\pi$ is 3-NA and 3^{rd}-order active secure.

4.4 χ permutation

For lanes, Equation (5), can be expressed as:

$$\chi(\mathbf{a}_{x,y}) = \mathbf{a}_{x,y} \oplus (\overline{\mathbf{a}_{x+1,y}} \wedge \mathbf{a}_{x+2,y}) \tag{29}$$

Accordingly, for $\mathbf{a}'_{x,y}$ we can use our impeccable AND gate f_{AND} from Section 3.5 and write

$$\chi'(\mathbf{a}'_{x,y}) = \mathbf{a}'_{x,y} \oplus f^{(16)}_{AND}(\overline{\mathbf{a}'_{x+1,y}}, \mathbf{a}'_{x+2,y}), \tag{30}$$

where $f^{(16)}_{AND}$ is a circuit of 16 f_{AND} gates to process a complete lane simultaneously.

Active Security of $\mathcal{F}_\chi$. In Section 3.5 we showed that the combination of an AND gate and f_{AND} is 3^{rd}-order active secure. For the inversion before and exclusive or operation after the AND gate it is obvious that errors do not accumulate. Therefore, $\mathcal{F}_\chi$ is 3-NA and 3^{rd}-order active secure.

4.5 ι permutation

For a lane $\mathbf{a}_{(x,y)}$, the ι permutation (Equation (6)) can be written as:

$$\iota(\mathbf{a}_{x,y}) = \begin{cases} \mathbf{a}_{x,y}, \text{ if } (x, y) = (0, 0) \\ \mathbf{a}_{x,y} + \mathbf{rc}_i : i \in [0, 24] \text{ otherwise} \end{cases} \tag{31}$$

To apply it to an encoded lane $\mathbf{a}'_{(x,y)} = \mathbf{a}_{(x,y)} \cdot \mathbf{P}$, the round constants $\mathbf{rc}_i : i \in [0, 24]$ must be encoded with $\mathbf{P}$. Therefore:

$$\iota'(\mathbf{a}'_{x,y}) = \begin{cases} \mathbf{a}'_{x,y}, \text{ if } (x, y) = (0, 0) \\ \mathbf{a}'_{x,y} + \mathbf{rc}_i \cdot \mathbf{P} = \mathbf{a}' + \mathbf{rc}'_i \text{ otherwise} \end{cases} \tag{32}$$

Active Security of $\mathcal{F}_\iota$. Since both ι and ι' perform only simple addition in $\mathbb{F}_{2^{64}}$, faulting a single bit corrupts exactly one output bit. Therefore, $\mathcal{F}_\iota$ is 3-NA and 3^{rd}-order of active secure.

4.6 Composition of Keccak Round

So far, we provided standalone 3^{rd}-order active security proofs for each of the five steps within a KECCAK round. As laid out in [DN20], this is not sufficient for the complete KECCAK round logic to be 3^{rd}-order active secure. In the following, we derive this proof step by step. We start with the combination of θ and ρ permutations.

Active Security of θ and ρ combination. A combination of $\mathcal{F}_\rho \circ \mathcal{F}_\theta$ can be represented by two functions:

$$F = \left(\mathbf{a}_{x,y} \oplus \bigoplus_{y=0}^{4} \mathbf{a}_{x-1,y} \oplus \left(\bigoplus_{y=0}^{4} \mathbf{a}_{x+1,y}\right) \cdot \mathbf{R}^{(1)}\right) \cdot \mathbf{R}^{(t)}$$
$$F' = \left(\mathbf{a}'_{x,y} \oplus \bigoplus_{y=0}^{4} \mathbf{a}'_{x-1,y} \oplus \left(\bigoplus_{y=0}^{4} \mathbf{a}'_{x+1,y}\right) \cdot \mathbf{T}^{(1)}\right) \cdot \mathbf{T}^{(t)}$$

As described above, it is obvious that an addition in a binary field does not accumulate errors. Therefore, only two vector-matrix multiplications are in the scope of the security analysis. The attacker is able to inject faults into input, intermediate and output variables of these multiplications. We can extend Equation (25) to cover $\mathcal{F}_\rho \circ \mathcal{F}_\theta$ as follows:

$$\mathbf{e} = \left(\mathbf{e}_1 \cdot \begin{bmatrix} \mathbf{R}^{(1)} & \underset{\sim}{\mathbf{0}} \\ \underset{\sim}{\mathbf{0}} & \widetilde{\mathbf{T}^{(1)}} \end{bmatrix} + \mathbf{e}_2\right) \cdot \begin{bmatrix} \mathbf{R}^{(t)} & \underset{\sim}{\mathbf{0}} \\ \underset{\sim}{\mathbf{0}} & \widetilde{\mathbf{T}^{(t)}} \end{bmatrix} + \mathbf{e}_3 \tag{33}$$

Similar to our analysis in Section 4.1, we constrain the faulty terms $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3 : HW(\mathbf{e}_1)+0 < HW(\mathbf{e}_2) + HW(\mathbf{e}_3) < 4$. The attack is successful, if a combination of errors exists for which $\forall \mathbf{e}^{(i)} \in \mathbf{e}\ \mathbf{e}^{(i)} \in \mathcal{C}$.

Our investigation yields that for all $3\binom{128}{3} + 9\binom{128}{2} + 18\binom{128}{1} = 6,293,376$ possible combinations of $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ for each of the possible t values, no such patterns exist. Hence, the combination $\mathcal{F}_\rho \circ \mathcal{F}_\theta$ is 3^{rd}-order active secure.

Active Security of $\mathcal{F}_\iota \circ \mathcal{F}_\chi \circ \mathcal{F}_\pi$. Since $\mathcal{F}_\pi$, $\mathcal{F}_\chi$ and $\mathcal{F}_\iota$ are 3-NA (and 3^{rd}-order active secure), their outputs cannot contain more errors than inputs. For $\mathcal{F}_\chi \circ \mathcal{F}_\pi$:

$$\mathbf{e}_{OUT} = \mathcal{F}_\chi(\mathcal{F}_\pi(\mathbf{e}_1), \mathbf{e}_2) + \mathbf{e}_3$$

such that $HW(\mathcal{F}_\chi(\mathcal{F}_\pi(\mathbf{e}_1), \mathbf{e}_2)) \leq HW(\mathbf{e}_1) + HW(\mathbf{e}_2)$ due to non-accumulation property. Therefore, $HW(\mathbf{e}_{OUT}) \leq HW(\mathbf{e}_1) + HW(\mathbf{e}_2) + HW(\mathbf{e}_3)$. Since the maximum Hamming weight of injected faults does not exceed three, $\mathcal{F}_\chi \circ \mathcal{F}_\pi$ is 3-NA and 3^{rd}-order active secure.

For $\mathcal{F}_\iota \circ \mathcal{F}_\chi \circ \mathcal{F}_\pi$,

$$\mathbf{e}_{OUT} = \mathcal{F}_\iota(\mathcal{F}_\chi(\mathcal{F}_\pi(\mathbf{e}_1), \mathbf{e}_2) + \mathbf{e}_3) + \mathbf{e}_4 = \mathcal{F}_\iota(\mathbf{e}'_3) + \mathbf{e}_4,$$

where $HW(\mathbf{e}'_3) \leq HW(\mathbf{e}_1) + HW(\mathbf{e}_2) + HW(\mathbf{e}_3)$ due to their non-accumulation. Since $\mathcal{F}_\iota$ is 3-NA as well, $HW(\mathbf{e}_{OUT}) \leq HW(\mathbf{e}'_3) + HW(\mathbf{e}_4) \leq HW(\mathbf{e}_1) + HW(\mathbf{e}_2) + HW(\mathbf{e}_3) + HW(\mathbf{e}_4)$. $\mathcal{F}_\iota \circ \mathcal{F}_\chi \circ \mathcal{F}_\pi$ is 3^{rd}-order active secure due to the fact that $HW(\mathbf{e}_1) + HW(\mathbf{e}_2) + HW(\mathbf{e}_3) + HW(\mathbf{e}_4) < 4$.

Active Security of Keccak Round. The complete KECCAK round can be expressed as $\mathcal{F}_\iota \circ \mathcal{F}_\chi \circ \mathcal{F}_\pi \circ \mathcal{F}_\rho \circ \mathcal{F}_\theta$. It is observable that the circuit consists of 2 parts, which are 3^{rd}-order active secure by their own. Due to the cyclic structure of KECCAK-$f[b]$, we can rewrite this equation to focus on the checkpoint: $\mathcal{F}_\pi \circ \mathcal{F}_\rho \circ \mathcal{F}_\theta \circ \mathcal{F}_\iota \circ \mathcal{F}_\chi$. It does not reflect the actual functionality of the circuit, but allows us to verify its security efficiently. Since π and π' permutations can be implemented via rewiring, the equation can be modified further:

$$\mathcal{F} = \mathcal{F}_\rho \circ \mathcal{F}_\theta \circ \mathcal{F}_\iota \circ \mathcal{F}_\chi \circ \mathcal{F}_\pi$$

Therefore, we divide $\mathcal{F}$ into $\mathcal{F}_2 \circ \mathcal{F}_1$, where $\mathcal{F}_2 = \mathcal{F}_\rho \circ \mathcal{F}_\theta$ and $\mathcal{F}_1 = \mathcal{F}_\iota \circ \mathcal{F}_\chi \circ \mathcal{F}_\pi$. So,

$$\mathbf{e}_{OUT} = \mathcal{F}_2(\mathcal{F}_1(\mathbf{e}_1), \mathbf{e}_2) + \mathbf{e}_3 = \mathcal{F}_2(\mathbf{e}'_1, \mathbf{e}_2) + \mathbf{e}_3,$$

where $\mathbf{e}'_1$, such that $HW(\mathbf{e}'_1) \leq HW(\mathbf{e}_1)$, is the input error of $\mathcal{F}_2$, while $\mathbf{e}_2$ can be distributed across multiple variables. However, $\mathcal{F}_2$ is proven to be secure if $HW(\mathbf{e}_1) + HW(\mathbf{e}_2) \leq 3$. Therefore, $HW(\mathbf{e}_{OUT}) < HW(\mathbf{e}_1) + HW(\mathbf{e}_2) + HW(\mathbf{e}_3)$. So, the KECCAK round is 3^{rd}-order active secure.

In the implementation, a state register is used to store intermediate round results. However, since the register simply transmits stored variables, it does not affect the security. It is also important to mention, that, of all five permutations, only the χ circuit has a checkpoint and accordingly an abort mechanism. This is possible due to the linearity of the remaining four permutations. Therefore, an error propagation within these permutations can be predicted strictly, which is not the case for non-linear circuits. However, the checkpoint protects the sole non-linear circuit $\mathcal{F}_\chi$, detects errors and aborts the operation, if required.

Remark 8. The checkpoint $\mathcal{S}$ is implemented between $\mathcal{F}_\pi$ and $\mathcal{F}_\chi$. Within the checkpoint is the encoding module with matrix $\mathbf{P}$. We reuse this to encode and decode data at the beginning and end of KECCAK-$f[b]$ (see Section 5. Therefore, the checkpoint is used in three cases: $\mathcal{F}_\pi \circ \mathcal{F}_\rho \circ \mathcal{F}_\theta$ (for the first round), $\mathcal{F}_\rho \circ \mathcal{F}_\theta \circ \mathcal{F}_\iota \circ \mathcal{F}_\chi \circ \mathcal{F}_\pi$ (for the main cycle) and $\mathcal{F}_\iota \circ \mathcal{F}_\chi$ (for the final check). The main loop is 3^{rd}-order active secure. Since $\mathcal{F}_\pi$ only changes lane indices, it can be removed from $\mathcal{F}_\pi \circ \mathcal{F}_\rho \circ \mathcal{F}_\theta$ and added to $\mathcal{F}_\iota \circ \mathcal{F}_\chi$. Therefore, we obtain circuits, which are described above and which are 3^{rd}-order active secure as well. So, the resulting KECCAK implementation is 3^{rd}-order active secure according to all possible data paths.

5 Implementation and Evaluation

In this section, we combine the five steps of a KECCAK round from Section 4 and the encoding/decoding matrix from Section 3.2 to one impeccable KECCAK. We evaluate our design for FPGAs and ASICs.

5.1 Impeccable Keccak Module

According to [AMR+20], we can combine the five permutations from Section 4 to $\mathcal{R}'$ as:

$$\mathcal{R}' = \iota' \circ \chi' \circ \pi' \circ \rho' \circ \theta' \tag{34}$$

Therefore, the KECCAK module consists of the two round functions $\mathcal{R}$ (Equation (1)) and $\mathcal{R}'$ (Equation (34)). In addition, the checkpoint is required before χ and χ' permutations to fulfill the requirements for 3^{rd}-order active security. As described in Section 3.2, the checkpoint performs decoding of the π' output and compares it with the result of the original π permutation. We use the matrix $\mathbf{P}'$, which we introduced in Equation (9), for

encoding and decoding (in the following, we denote it as $\mathbf{P}$). In terms of hardware design, this means that we can re-use the same logic for en- and decoding. This introduces the constraint that encoding and decoding of different data (blocks) can not be performed simultaneously.

Our KECCAK implementation is depicted in Figure 6. After the π and π' permutation, the checkpoint $\mathcal{S}$ is instantiated. It consists of a module that corresponds to a vector-matrix multiplication with $\mathbf{P}$ and the comparator, which produces an error signal s, if an error is detected. The KECCAK and ENCODED KECCAK modules each have a dedicated state register. It should be noted that we view the integrity of the $Keccak_{IN}$, $Keccak_{OUT}$ and s signals as out of scope for this work. A generic interface for our impeccable KECCAK could be a standard FIFO or shift register which allows to store integrity bits for each word. In this case it is important that the implementation manages a "secure hand-off", e.g. by checking both the integrity of the data word and its redundancy that were written into the state registers and the consistency of the data word with its integrity bits, before abandoning the integrity bits. Since $\mathbf{P}$ can be reused during the encoding and the decoding, the module requires one clock cycle to encode the KECCAK input and store it in the corresponding register. Another clock cycle is required to decode the KECCAK output using the same module $\mathbf{P}$. A single round of KECCAK-$f[b]$ is computed in one clock cycle. Therefore, an execution of the KECCAK-$f[1600]$ function without data transmissions takes 26 clock cycles.

5.2 Evaluation

We implemented our impeccable KECCAK in VHDL and synthesized it for Artix-7 FPGAs[2] and the generic FreePDK45[3] open-source standard-cell library[4]. The results and a comparison with related work are presented in Table 1 and Table 2.

5.2.1 Evaluation in a bit-flip Fault Model

In Section 1.1, we introduced other KECCAK designs. In this section, we revisit these designs and their security claims for a thorough comparison with our work. In general, the designs can be classified to be either area/instance- or time-redundant. We evaluate all designs according to the definitions of a fault and active security from Section 3.1. It should be noted that this fault and security model does not necessarily match the physical reality of fault injections. In particular for time-redundant designs, permanent faults might be an issue. Further, when the physical realities of chip design are taken into account, it becomes obvious that a single bit-flip - e.g. within a clock tree - can lead to multiple bit-flips. Such effects are obviously not covered within the bit/variable fault model in Definition 1 and Definition 2. Including such weaknesses into the evaluation is difficult. Even frameworks such as [RBFSG22, NOV+22, RBRSS+21] conduct their analysis after RTL synthesis, i.e. on netlist level, therefore any structures that are only created during place-and-route can not be investigated. We discuss our design with relation to real-world fault attacks in the next section.

Comparison with State-of-the-Art Designs. The predictor-compressor design by Luo et al. [LLF16] was already investigated in Section 1.1. Its resilience is based on one- or multiple dimensional parity checks. It was shown that - for one dimensional parity checks - two precise bit-flips are always sufficient to compromise the design. Further, there seems to be the possibility to corrupt the design at a short period of time with a single bit-flip.

[2] using Vivado 2020.2 and the XC7A35T-1CPG236C part
[3] https://eda.ncsu.edu/freepdk/freepdk45/
[4] using Cadence Genus, Version 21.10-p002_1

Table 1: Synthesis results for FPGAs.

Design	FPGA	Keccak (no countermeasures)				Keccak (with countermeasures)			
		Slices	LUT	FF	Frequency (MHz)	Slices	LUT	FF	Frequency (MHz)
[MB23]	Virtex-5	1,370	-	-	258.6	1,680	-	-	387
[MBM21]	Virtex-5	1,350	-	-	252.4	1,601	-	-	365.2
[KMBM17]	Virtex-5	1,356	-	-	296.5	2,260	-	-	291.3
[TAMRAB+22]-HC	Virtex-7	-	2,453	2,457	194.3	-	28,702	18,256	41.1
[TAMRAB+22]-TMR-HC	Virtex-7	-	2,453	2,457	194.3	-	27,233	26,261	59
This work	Artix-7	1,036	3,953	1,621	222.2	3,269	12,408	3,260	111.1

Table 2: Synthesis results for ASICs.

Design	PDK	Keccak (no countermeasures)			Keccak (with countermeasures)		
		Chip Area (μm^2)	Gate Eqv. (kG)	Frequency (MHz)	Chip Area (μm^2)	Gate Eqv. (kG)	Frequency (MHz)
[BSMKRM14]	Synopsys 65nm	66,306	47.0	676	69,249	49.1	1,192
[LLF16]	FreePDK45 45nm	41,612	-	256.9	66,621	-	228.3
[PADM19]-BLAZE	FreePDK45 45nm	-	-	-	-	2,495	-
[PADM19]-FAST	FreePDK45 45nm	-	-	-	-	1,318	-
[PADM19]-FUR	FreePDK45 45nm	-	-	-	-	177.2	-
[PADM19]-KIT	FreePDK45 45nm	-	-	-	-	89.8	-
This work	FreePDK45 45nm	45,569	57.1	1,316	142,782	177.7	719.4

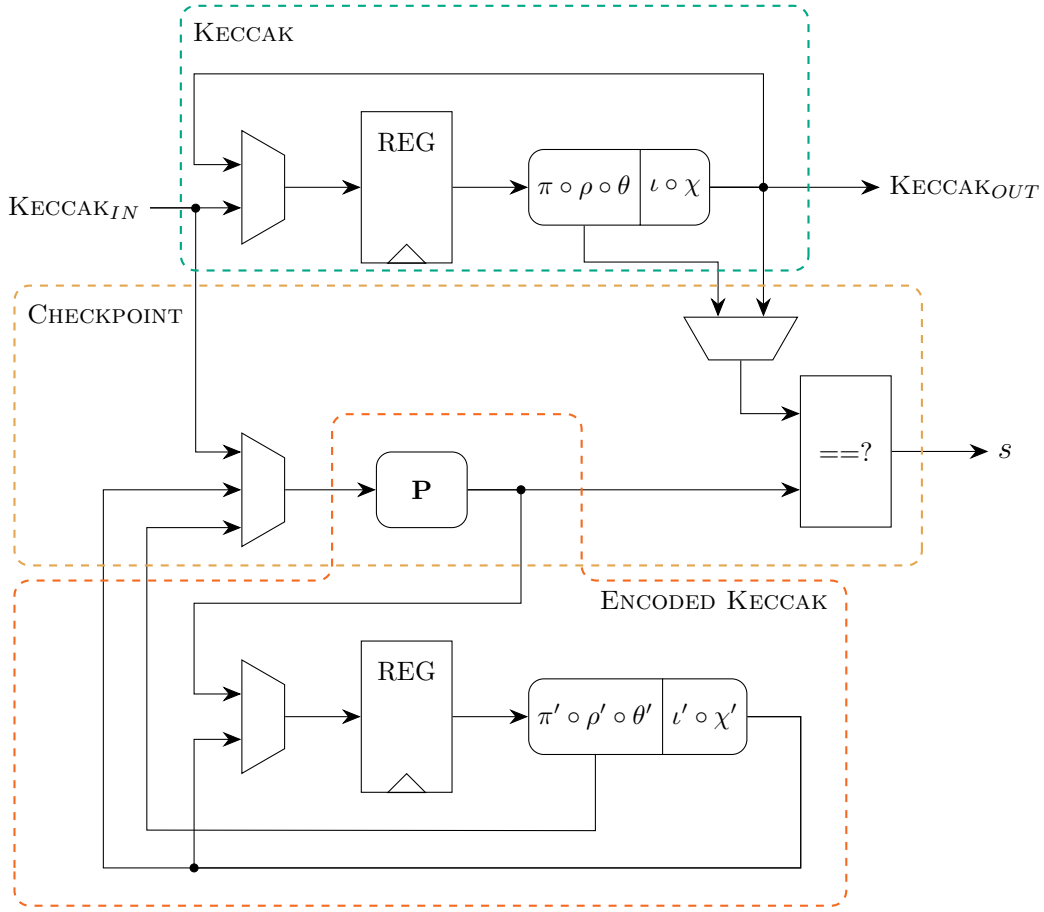


Figure 6: The impeccable Keccak implementation. It consists of three parts: the original Keccak module (highlighted with green color), the Encoded Keccak module (highlighted with red color) and the checkpoint (highlighted with yellow color).

Using the notion from [DN20], this design would then not even meet 1^{st}-order of active security. At an overhead of factor 1.6 for the chip area, their solution is relatively compact.

Torres-Alvarado et al. [TAMRAB+22] combined Hamming codes with triple redundancy. More concretely, the authors proposed two designs with error detection capabilities, [TAMRAB+22]-HC and [TAMRAB+22]-TMR-HC in Table 1. The [TAMRAB+22]-HC design only protects registers by encoding the Keccak state with a Hamming code. [TAMRAB+22]-TMR-HC uses additional triple-redundant permutation modules to also protect combinatorial logic. The outputs of the Keccak instances are fed into a voting system, which corrects errors based on a majority vote. Therefore, if two precise faults are introduced into two of the three inputs, the attacker can successfully compromise the implementation. Further, similar to [LLF16], the data could be corrupted if a single fault is injected exactly between the voting system and the register. Therefore, this design is not even 1^{st}-order active secure. Since both implementations use an error correction scheme, the comparison with our approach is not fair. However, it is interesting to note the difference between designs developed for safety and reliability versus designs developed for security. Table 1 shows that the designs from [TAMRAB+22] are rather costly.

Hassen et al. [MBM21] described a Keccak implementation, where the same data is processed twice using pipelining. Intermediate results are stored in registers and compared with an offset of one cycle. Kehri et al. [KMBM17] duplicated the Keccak round logic and

introduced a scrambling of the state before storing it in the according registers. Further, Hassen et al. [MB23] combined the pipelined execution from [MBM21] with the state shuffling from [KMBM17]. From their descriptions and evaluations we can conclude that the design in [KMBM17] is 1^{st}-order active secure. Its shuffled state registers might complicate differential fault analysis, but not the grafting trees attack. The designs in [MB23, MBM21] seem to be 1^{st}-order active secure. In comparison to [MBM21], [MB23] makes precise fault injection into registers more difficult. In [MB23], probabilities of 0.023 % and 0.014 % for 2-bit (3-bit errors) to pass undetected are derived from simulation. All three methods can be implemented efficiently (Table 1). The overhead of [KMBM17] is smaller than factor two, since only combinatorial logic is duplicated. [MBM21, MB23] are also compact since they combine area- and time-redundancy. It is noteworthy that the protected and pipelined designs in [MB23, KMBM17] achieve higher frequencies than the unprotected design they used for reference.

The authors of [BSMKRM14, LZFD16] also use shuffling of the KECCAK state for their countermeasures. The rotation invariance of the KECCAK is exploited by rotating the state randomly for each KECCAK-$f[b]$ permutation. The KECCAK-$f[b]$ function is executed multiple times, i.e. time-redundant, and results are compared after each run. Assuming that an adversary is able to inject bit-flips with a high spatial and temporal resolution, it is not guaranteed that the same bit of the state is hit. There are 64 possibilities to rotate the state (including rotation by 0), which gives a probability of $\frac{1}{64}$ that the bit is located in the same position. Depending on the adversarial model, the developers can decide to repeat the permutation two or more times. Since the design is entirely time-redundant and only needs additional shuffling logic, the area overhead of [BSMKRM14] is only 4.3 %. Unfortunately, the authors of [LZFD16] did not provide implementation results. Due to its time-redundancy, the approach also has a severe latency penalty.

In [PADM19], Purnal et al. use the CAPA approach [RDMB+18] to protect a KECCAK against side-channel and fault attacks. CAPA builds on a *tile-probe-and-fault* model and employs information-theoretic unpredictable MAC tags to mitigate fault injection. Each sensitive value is accompanied by m tags. Compared to all previous approaches, CAPA allows a formal verification of the countermeasure. However, there is no 100 % probability that any faults are detected. The detection probability is given as $1 - 2^{-m}$. As such, the design is not even 1^{st}-active secure. It should be noted, that at the time, this notion did not yet exist. Instead, the authors provided a experimental verification of the detection probability for their KECCAK design. The values in Table 2 are for $m = 2$, i.e. a fault detection probability of 75 %, and second-order masking. The authors provide four design trade-offs, of which only BLAZE is completely parallel. Due to the masking countermeasure, the area consumption is rather high.

Compared to the high-speed KECCAK hardware implementation of the KECCAK Team [BDPvA11], our design has a significant area overhead: factor 3.2 on the FPGA and factor 3.1 in ASIC chip area. In addition, the maximum frequency is reduced by half due to additional encoding and decoding operations. In Section 4, we proved that our design is 3^{rd}-order active secure. More generally, our design can be broken only with $n \geq 4$ bit-flips. If these bit-flips are assumed to be random, the following probabilities regarding a successful attack hold:

- bit-flips spread over the complete KECCAK state: $< 1.5 \cdot 10^{-9}$
- bit-flips within a lane in the KECCAK state: $< 2.4 \cdot 10^{-5}$
- bit-flips within an 8-bit code-word (i.e. 4-bit data, 4-bit redundancy): < 0.25

A success probability of 100 % is only achievable, if the attacker knows the processed data and can manipulate at least four bits precisely. To identify potential bits, or so called points of interest, the attacker can use different techniques. Detailed knowledge of the

layout of the integrated circuit, e.g. obtained via reverse engineering, could help to identify relevant positions. However, the adversary still needs information which data is processed when. While this knowledge could be obtianed via side-channel analysis, it makes the attack highly complex. Even with the knowledge of the circuit, the processed data, and the timing, an adversary needs four precise single-bit errors.

We emphasize that our design does not protect Keccak against statistical ineffective fault analysis (SIFA) [DEK+18]. However, due to the urgent need to thwart the grafting trees attack, we leave protection against SIFA for future research.

5.2.2 Evaluation in real-world Fault Models

As mentioned in the previous section, mapping security in a bit-flip model (as e.g. the notion of active security from [DN20]) to resilience against real-world fault attacks is hard. This is, because the placement and routing of an impeccable circuit is unknown from its netlist and depends on the hardware design toolchain. Further, the grafting trees attack does not require a precise number of flipped bits, but only an incorrect output that passes any error detection countermeasures. Thus, imprecise faults that affect many bits can also lead to success for an adversary. However, we argue that everything but precise attacks are unrealistic. On the one hand, we base this on the practical evaluation of complete redundant impeccable circuits in the work of Bartkewitz et al. [BBM+22] (see below). On the other hand, our intuition is as follows: Each of the 400 8-bit code-words (4-bit original data, 4-bit redundancy) in our design is either affected or unaffected by a fault. If it is affected, all combinations, where three or less bits are faulty will be detected with a probability of 100 %. As described above, for $n \geq 4$ random bit-flips, the probability that the code-word is valid is at most 25 %. For imprecise fault attacks, we estimate the probability that this condition applies to all affected code words to be sufficiently low to focus our further analysis on an adversarial model, where the attacker attempts to insert precise and localized faults, which are limited to as few code-words as possible. Nevertheless, we think that an elaborate analysis (similar to [BBM+22]) of countermeasures based on encoded data, i.e. including both simple memory elements featuring code-based checks such as [TAMRAB+22, LLF16] and impeccable circuit designs such as ours, with different precise and imprecise fault injection methods, is an important analysis for future work in this direction.

The layout of the integrated circuit and its technology node determine how hard it is to insert faults into multiple bits of interest with a single glitch (e.g. clock or supply voltage) or pulse (e.g. laser or electromagnetic). The results from [DBC+18] offer some insight regarding the capability of laser fault injection, arguably the most precise method for fault attacks, independent from the design. For an 28 nm CMOS node, the authors demonstrate that 1-bit faults are feasible. Further, their results indicate that, while multi-bit faults are possible, the number of flipped bits is hard to control and they are less likely. Furthermore, there are differences between setting and resetting a bit.

Bartkewitz et al. provide further insights into the resilience of impeccable circuits in practice. [BBM+22]. They use various hardware implementations of the SKINNY [BJK+16] cipher. In particular, they compare full-redundant impaccable circuits, where the redundant data is processed independent from the actual data, with encoded circuits, where the redundant data is insufficient for independent processing. In this case, a dependency in the data flow of the circuit processing the actual data and the circuit processing the redundant data must be introduced. Bartkewitz et al. show, that circuits with this insufficient redundancy can be successfully faulted with laser fault injection. The only designs they were not able to compromise, are a simple circuit, where the original design is duplicated and the outputs are compared and an impeccable circuit with complete redundancy ([8,4,4] coding scheme), similar to our Impeccable Keccak. In this case, the simple duplicated design might seem more attractive, as it is 27 % smaller than the encoded circuit. Considering

our results for Keccak without countermeasures (Table 2), we estimate that our Keccak design has a similar overhead, compared to simple duplication (the duplication requires two Keccak instances and a comparator). However, in [BBM+22], the authors only consider attacks with a single laser beam. In [CGV+22], it was demonstrated that up to four bits can be flipped with four synchronous laser beams. We suspect that it is simpler to introduce two symmetric (target the same bit in both instances) faults with two laser beams than flipping four bits, such that our design does not detect the fault. As described above, depending on the technology and layout of the integrated circuit, an adversary does not necessarily need four laser beams to achieve this. However, in [BBM+22], the authors were unable to achieve this with a single laser beam.

It is worth noting, that protection against SIFA could make protected designs based on redundant instances more favorable. To protect against SIFA, one could either use simple triplication or quadruplication in combination with a majority voting system that suppresses faults. For a design based on encoded data, error correction instead of detection would be required. As stated above, we emphasize the importance of protection against SIFA for future works.

5.3 Towards a Secure SPHINCS+ Implementation

Our motivation for designing an impeccable Keccak was to protect SPHINCS+ from the grafting trees attack [CMP18]. In general, secure implementations must be protected against side-channel and fault attacks. In the case of SPHINCS+, side-channel attacks are less of a concern. This is because a side-channel attack on SPHINCS+ must target the underlying hash function [KGB+18], which is relatively hard, even for unprotected implementations of Keccak, due to its sponge construction [DEM+17] and the completely parallel permutation. Fault attacks, such as the grafting trees attack [CMP18], are a much more severe risk for the implementation security of SPHINCS+. Therefore, we built on the impeccable circuits approach, instead of using other work, which combines masking with resilience against fault attacks [SMG16, DN20, FGM+23, DN21, BEF+23, FRBSG22, DMAN+18, RDMB+18]. This allows us to build a performant and cost efficient impeccable Keccak.

Further, we mainly investigated the protection of Keccak in this work. This is because we view the protection of other building blocks in SPHINCS+ as mostly solved. Hashing is by far the most time-consuming operation in SPHINCS+, the remaining time is mostly spent with managing hashchains or Merkle trees. While the attack surface of the grafting trees attack expands to these operations as well, it is easy to implement them on a CPU and harden the CPU with generic countermeasures. One might argue that the hash algorithm could also be implemented on the CPU. However, since most of SPHINCS+'s computation time is spent on the Keccak function, its performance can be significantly improved by using a hardware accelerator for this operation.

Therefore, we recommend to implement SPHINCS+ on a microcontroller that includes countermeasures against fault injection and uses our impeccable Keccak as co-processor. In fact, with the OpenTitan[5], a suitable system-on-chip already exists. Its countermeasures include (among others): generic hardening of state machines and control signals with sparse encodings, one-hot encodings, scrambled memories with error correction codes, a lockstep core for the main processor, bus integrity checking, register file with error correction codes and a hardened program counter in the main processor. Obviously, the application of these countermeasures to a complex system like a CPU can not be evaluated under the same formal model of active security as our design. Therefore, their effectiveness must be evaluated manually with heuristic simulations or laboratory investigations.

The OpenTitan also includes a Keccak core, which is protected against side-channel

[5] https://github.com/lowRISC/opentitan

attacks. Due to the countermeasures listed above, localized fault attacks by means of electromagnetic or laser fault injection on the Keccak core would be the most promising approach to attack SPHINCS$^+$ if it were implemented on the OpenTitan. If our design were integrated into the OpenTitan, all operations within the SPHINCS$^+$ algorithm would be hardened against fault injection at almost no latency penalty.

6 Conclusion

In this paper we combined the concepts of impeccable circuits [AMR+20, SRM20, RSM21] and active security [DN20] to propose an impeccable Keccak. Our design is provably 3^{rd}-order active secure, i.e. an adversary needs to flip at least four bits. In this context, a bit refers to either a single flip-flop or a wire. Further, not any four bits in the Keccak state lead to a successful attack. With an area overhead of factor 3.2 and 3.1 on FPGA and ASIC, a decrease of the maximum frequency by 50 % and practically no overhead in cycle counts, our design is more than competitive with state-of-the-art work on reliable Keccak implementations. Besides, we laid out that state-of-the-art proposals - some of which were designed for safety rather than security - show flaws, when the formal notion of active security is used. We argue that these flaws, e.g. leaving data unprotected for a short moment in time, could be exploited with highly precise fault injection. With our design, the adversary needs to flip four or more bits at any given time.

Due to the powerful grafting trees attack, SPHINCS$^+$ implementations are in dire need of fault injection countermeasures. By integrating our design into a microcontroller that incorporates generic fault injection countermeasures, the vulnerability to fault injection attacks is massively reduced to a point where an attack is hardly feasible.

While developing our impeccable Keccak, we investigated how the resilience of impeccable circuits can be verified. For this, we proposed a model that captures arbitrary fault attacks on impeccable circuits. Further, we combined this with the notion of active security [DN20] and demonstrated that this exposes unexpected weaknesses in previous work [SMG16]. We showed how the security of arbitrary linear and non-linear impeccable circuits can be proven. Further, we derived a 3^{rd}-order strong-non-accumulative AND gate that we use to implement the χ permutation in Keccak, but is also of general interest.

Future work could investigate how our design compares to a Keccak implementation with combined countermeasures, i.e. against passive side-channel and active fault injection attacks (e.g. SIFA). Moreover, we see potential value in adapting the ongoing research on verification [RBFSG22, RBRSS+21, AWMN20, NOV+22] for our evaluation methodology of impeccable circuits and verifying our design with these approaches. Finally, we also want to spark research on formal security notions and adversarial models that are more closely related to the physical reality of fault injection attacks. The random fault model [DN22] is a step in this direction. Since the proposed design does not prevent SIFA, the protection against it is an important step towards secure Keccak and SPHINCS$^+$ implementations. Future research could also be connected to finding efficient coding schemes with greater distances (e.g. [16,8,5] Hamming code) or alternative coding schemes (e.g. low or moderate-density parity check codes), building impeccable circuits from them and evaluating them in the random fault model.

References

[AAC+22] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process. National Institute of Standards and Technology Interagency or Internal Report, NIST IR 8413-upd1, 2022. https://doi.org/10.6028/NIST.IR.8413-upd1.

[ABB+22] Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, and Bas Westerbaan. SPHINCS+. Submission to the NIST post-quantum project, v.3.1, 2022. https://sphincs.org/resources.html.

[ALCZ20] Dorian Amiet, Lukas Leuenberger, Andreas Curiger, and Paul Zbinden. FPGA-based SPHINCS+ Implementations: Mind the Glitch. In *2020 23rd Euromicro Conference on Digital System Design (DSD)*, pages 229–237, 2020.

[AMR+20] Anita Aghaie, Amir Moradi, Shahram Rasoolzadeh, Aein Rezaei Shahmirzadi, Falk Schellenberg, and Tobias Schneider. Impeccable Circuits. *IEEE Transactions on Computers*, 69(3):361–376, 2020.

[AWMN20] Victor Arribas, Felix Wegener, Amir Moradi, and Svetla Nikova. Cryptographic Fault Diagnosis using VerFI. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 229–240, 2020.

[BBM+22] Timo Bartkewitz, Sven Bettendorf, Thorben Moos, Amir Moradi, and Falk Schellenberg. Beware of Insufficient Redundancy: An Experimental Evaluation of Code-based FI Countermeasures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(3):438–462, Jun. 2022.

[BDK+07] Johannes Buchmann, Erik Dahmen, Elena Klintsevich, Katsuyuki Okeya, and Camille Vuillaume. Merkle Signatures with Virtually Unlimited Signature Capacity. In *Applied Cryptography and Network Security*, pages 31–45, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[BDPvA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles van Asche. The Keccak reference. https://keccak.team/files/Keccak-reference-3.0.pdf, 2011.

[BEF+23] Sebastian Berndt, Thomas Eisenbarth, Sebastian Faust, Marc Gourjon, Maximilian Orlt, and Okan Seker. Combined Fault and Leakage Resilience: Composability, Constructions and Compiler. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 377–409, Cham, 2023. Springer Nature Switzerland.

[BGS15] Nasour Bagheri, Navid Ghaedi, and Somitra Kumar Sanadhya. Differential Fault Analysis of SHA-3. In Alex Biryukov and Vipul Goyal, editors, *Progress in Cryptology – INDOCRYPT 2015*, pages 253–269, Cham, 2015. Springer International Publishing.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016*, pages 123–153, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[BKL+07] 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 Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, pages 450–466, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[BSMKRM14] Siavash Bayat-Sarmadi, Mehran Mozaffari-Kermani, and Arash Reyhani-Masoleh. Efficient and Concurrent Reliable Realization of the Secure Cryptographic SHA-3 Algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(7):1105–1109, 2014.

[CGV+22] Brice Colombier, Paul Grandamme, Julien Vernay, Émilie Chanavat, Lilian Bossuet, Lucie de Laulanié, and Bruno Chassagne. Multi-Spot Laser Fault Injection Setup: New Possibilities for Fault Injection Attacks. In Vincent Grosso and Thomas Pöppelmann, editors, *Smart Card Research and Advanced Applications*, pages 151–166, Cham, 2022. Springer International Publishing.

[CMP18] Laurent Castelnovi, Ange Martinelli, and Thomas Prest. Grafting Trees: A Fault Attack Against the SPHINCS Framework. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography*, pages 165–184, Cham, 2018. Springer International Publishing.

[DBC+18] Jean-Max Dutertre, Vincent Beroulle, Philippe Candelier, Stephan De Castro, Louis-Barthelemy Faber, Marie-Lise Flottes, Philippe Gendrier, David Hély, Regis Leveugle, Paolo Maistri, Giorgio Di Natale, Athanasios Papadimitriou, and Bruno Rouzeyre. Laser Fault Injection at the CMOS 28 nm Technology Node: an Analysis of the Fault Model. In *2018 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 1–6, 2018.

[DEK+18] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. SIFA: Exploiting Ineffective Fault Inductions on Symmetric Cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):547–572, Aug. 2018.

[DEM+17] Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, and Thomas Unterluggauer. ISAP – Towards Side-Channel Secure Authenticated Encryption. *IACR Transactions on Symmetric Cryptology*, 2017(1):80–105, Mar. 2017.

[DMAN+18] Lauren De Meyer, Victor Arribas, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. M& M: Masks and Macs against Physical Attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(1):25–50, Nov. 2018.

[DN20] Siemen Dhooghe and Svetla Nikova. My Gadget Just Cares for Me - How NINA Can Prove Security Against Combined Attacks. In Stanislaw

Jarecki, editor, *Topics in Cryptology – CT-RSA 2020*, pages 35–55, Cham, 2020. Springer International Publishing.

[DN21] Siemen Dhooghe and Svetla Nikova. Let's Tessellate: Tiling for Security Against Advanced Probe and Fault Adversaries. In Pierre-Yvan Liardet and Nele Mentens, editors, *Smart Card Research and Advanced Applications*, pages 181–195, Cham, 2021. Springer International Publishing.

[DN22] Siemen Dhooghe and Svetla Nikova. The Random Fault Model. Cryptology ePrint Archive, Paper 2022/1627, 2022. https://eprint.iacr.org/2022/1627.

[EAB+23] Mohamed ElGhamrawy, Melissa Azouaoui, Olivier Bronchain, Joost Renes, Tobias Schneider, Markus Schönauer, Okan Seker, and Christine van Vredendaal. From MLWE to RLWE: A Differential Fault Attack on Randomized and Deterministic Dilithium. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(4):262–286, Aug. 2023.

[FGM+23] Jakob Feldtkeller, Tim Güneysu, Thorben Moos, Jan Richter-Brockmann, Sayandeep Saha, Pascal Sasdrich, and François-Xavier Standaert. Combined Private Circuits - Combined Security Refurbished. Cryptology ePrint Archive, Paper 2023/1341, 2023. https://eprint.iacr.org/2023/1341.

[FRBSG22] Jakob Feldtkeller, Jan Richter-Brockmann, Pascal Sasdrich, and Tim Güneysu. CINI MINIS: Domain Isolation for Fault and Combined Security. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 1023–1036, New York, NY, USA, 2022. Association for Computing Machinery.

[GBH18] Leon Groot Bruinderink and Andreas Hülsing. "Oops, I Did It Again" – Security of One-Time Signatures Under Two-Message Attacks. In *Selected Areas in Cryptography – SAC 2017*, pages 299–322. Springer International Publishing, 2018.

[GBP18] Leon Groot Bruinderink and Peter Pessl. Differential Fault Attacks on Deterministic Lattice Signatures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):21–43, Aug. 2018.

[Gen23] Aymeric Genêt. On protecting SPHINCS+ against fault attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(2):80–114, Mar. 2023.

[GKPM18] Aymeric Genêt, Matthias J. Kannwischer, Hervé Pelletier, and Andrew McLauchlan. Practical Fault Injection Attacks on SPHINCS. Cryptology ePrint Archive, Paper 2018/674, 2018. https://eprint.iacr.org/2018/674.

[Gol87] Oded Goldreich. Two Remarks Concerning the Goldwasser-Micali-Rivest Signature Scheme. In *Proceedings on Advances in Cryptology—CRYPTO '86*, page 104–110, Berlin, Heidelberg, 1987. Springer-Verlag.

[HRB13] Andreas Hülsing, Lea Rausch, and Johannes Buchmann. Optimal Parameters for XMSSMT. In Alfredo Cuzzocrea, Christian Kittl, Dimitris E. Simos, Edgar Weippl, and Lida Xu, editors, *Security Engineering and Intelligence Informatics*, pages 194–208, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[KGB+18] Matthias J. Kannwischer, Aymeric Genêt, Denis Butin, Juliane Krämer, and Johannes Buchmann. Differential Power Analysis of XMSS and SPHINCS. In Junfeng Fan and Benedikt Gierlichs, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 168–188, Cham, 2018. Springer International Publishing.

[KMBM17] Fatma Kahri, Hassen Mestiri, Belgacem Bouallegue, and Mohsen Machhout. Fault Attacks Resistant Architecture for KECCAK Hash Function. *International Journal of Advanced Computer Science and Applications*, 8(5), 2017.

[LAFW17] Pei Luo, Konstantinos Athanasiou, Yunsi Fei, and Thomas Wahl. Algebraic fault analysis of SHA-3. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 151–156, 2017.

[LLF16] Pei Luo, Cheng Li, and Yunsi Fei. Concurrent error detection for reliable SHA-3 design. In *2016 International Great Lakes Symposium on VLSI (GLSVLSI)*, pages 39–44, 2016.

[LZFD16] Pei Luo, Liwei Zhang, Yunsi Fei, and A. Adam Ding. An Improvement of Both Security and Reliability for Keccak Implementations on Smart Card. Cryptology ePrint Archive, Paper 2016/214, 2016. https://eprint.iacr.org/2016/214.

[MB23] Hassen Mestiri and Imen Barraj. High-Speed Hardware Architecture Based on Error Detection for KECCAK. *Micromachines*, 14(6), 2023.

[MBM21] Hassen Mestiri, Imen Barraj, and Mohsen Machhout. Analysis and Detection of Errors in KECCAK Hardware Implementation. In *2021 IEEE International Conference on Design and Test of Integrated Micro and Nano-Systems (DTS)*, pages 1–6, 2021.

[NIS15] Information Technology Laborartory NIST. FIPS 202 Standard - Permutation-Based Hash and Extendable-Output Functions. Federal Information Processing Standards Publication, 2015. https://doi.org/10.6028/NIST.FIPS.202.

[NIS23] Information Technology Laborartory NIST. FIPS 205 (Draft) - Stateless Hash-Based Digital Signature Standard. Federal Information Processing Standards Publication, 2023. https://doi.org/10.6028/NIST.FIPS.205.ipd.

[NOV+22] Pascal Nasahl, Miguel Osorio, Pirmin Vogel, Michael Schaffner, Timothy Trippel, Dominic Rizzo, and Stefan Mangard. SYNFI: Pre-Silicon Fault Analysis of an Open-Source Secure Element. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):56–87, Aug. 2022.

[PADM19] Antoon Purnal, Victor Arribas, and Lauren De Meyer. Trade-offs in Protecting Keccak Against Combined Side-Channel and Fault Attacks. In Ilia Polian and Marc Stöttinger, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 285–302, Cham, 2019. Springer International Publishing.

[RBFSG22] Jan Richter-Brockmann, Jakob Feldtkeller, Pascal Sasdrich, and Tim Güneysu. VERICA - Verification of Combined Attacks: Automated formal verification of security against simultaneous information leakage and tampering. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):255–284, Aug. 2022.

[RBRSS+21] Jan Richter-Brockmann, Aein Rezaei Shahmirzadi, Pascal Sasdrich, Amir Moradi, and Tim Güneysu. FIVER – Robust Verification of Countermeasures against Fault Injections. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4):447–473, Aug. 2021.

[RBSBG20] Jan Richter-Brockmann, Pascal Sasdrich, Florian Bache, and Tim Güneysu. Concurrent Error Detection Revisited: Hardware Protection against Fault and Side-Channel Attacks. ARES '20, New York, NY, USA, 2020. Association for Computing Machinery.

[RBSG23] Jan Richter-Brockmann, Pascal Sasdrich, and Tim Güneysu. Revisiting Fault Adversary Models – Hardware Faults in Theory and Practice. *IEEE Transactions on Computers*, 72(2):572–585, 2023.

[RDMB+18] Oscar Reparaz, Lauren De Meyer, Begül Bilgin, Victor Arribas, Svetla Nikova, Ventzislav Nikov, and Nigel Smart. CAPA: The Spirit of Beaver Against Physical Attacks. In *Advances in Cryptology – CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I*, page 121–151, Berlin, Heidelberg, 2018. Springer-Verlag.

[RSM21] Shahram Rasoolzadeh, Aein Rezaei Shahmirzadi, and Amir Moradi. Impeccable Circuits III. In *2021 IEEE International Test Conference (ITC)*, pages 163–169, 2021.

[SMG16] Tobias Schneider, Amir Moradi, and Tim Güneysu. ParTI – Towards Combined Hardware Countermeasures Against Side-Channel and Fault-Injection Attacks. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016*, pages 302–332, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[SRM20] Aein Rezaei Shahmirzadi, Shahram Rasoolzadeh, and Amir Moradi. Impeccable Circuits II. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.

[TAMRAB+22] Alan Torres-Alvarado, Luis Alberto Morales-Rosales, Ignacio Algredo-Badillo, Francisco López-Huerta, Mariana Lobato-Báez, and Juan Carlos López-Pimentel. An SHA-3 Hardware Architecture against Failures Based on Hamming Codes and Triple Modular Redundancy. *Sensors*, 22(8), 2022.

[WWO+23] Alexander Wagner, Vera Wesselkamp, Felix Oberhansl, Marc Schink, and Emanuele Strieder. Faulting Winternitz One-Time Signatures to Forge LMS, XMSS, or SPHINCS+ Signatures. In *Post-Quantum Cryptography, 2023*, pages 658–687. Springer Nature Switzerland, 2023.