Impeccable Circuits

Anita Aghaie[1], Amir Moradi[1], Shahram Rasoolzadeh[1],
Falk Schellenberg[1], and Tobias Schneider[2]*

[1] Horst Görtz Institute for IT Security, Ruhr-Universität Bochum, Germany
[2] ICTEAM/ELEN/Crypto Group, Université catholique de Louvain, Belgium
[1]{firstname.lastname}@rub.de [2]{firstname.lastname}@uclouvain.be

Abstract. Active physical attacks pose a serious threat to cryptographic hardware, i.e., by injecting faults during the computation. The tools to inject such faults have evolved over the last years and are becoming increasingly powerful. A promising approach to thwart this type of attacks is employing Concurrent Error Detection (CED) schemes. They are usually based on an Error-Detecting Code (EDC) which provides the capability to detect certain injected faults. Depending on the assumed adversary model, the potency of the CED scheme can be adapted during the design phase by adjusting the underlying code.

In this work, we propose a methodology to enable a correct, practical, and robust implementation of code-based CED schemes. Indeed, we show that a straightforward hardware implementation of a given code-based CED scheme very likely suffers from severe vulnerabilities and does not provide the desired level of protection against fault attacks. In particular, the propagation of faults into the combinatorial logic is often not considered in the security evaluation of these schemes. First, we formally define this detrimental effect and demonstrate its impact on the security of common CED schemes. Second, we introduce an implementation strategy to limit the negative effect of fault propagation. Third, in contrast to many other works where the fault coverage of an implementation equipped with an EDC is considered, we present a detailed implementation strategy which – based on the specification of the underlying EDC – can guarantee (i.e., 100% coverage rate) the detection of any fault. Fitting to the defined adversary model, this holds for any time of the computation and any location of the circuit – both in the data processing and in the control part. In short, we provide practical guidelines how to construct efficient CED schemes with arbitrary EDCs to achieve the desired level of protection against fault attacks. We evaluate the efficiency of our methodology in a case study considering several symmetric block ciphers (i.e., PRESENT, Skinny, Midori, GIFT, LED, and SIMON) for different design architectures and various linear EDCs with different fault detection capabilities.

1 Introduction

Small embedded devices are ubiquitous and receive particular attention in the Internet of Things (IoT). Often, such devices are expected to fulfill security relevant services like authentication or storage of private and sensitive data. The crux of the matter is the embedded device being in the hand of a potential attacker. This enables all sorts of physical attacks on the implementation of some cryptographic scheme, independent of their mathematical security. The goal of our work is to protect a circuit against one class of such attacks: fault attacks or active implementation attacks, first introduced by Boneh et al. [17]. Here, the attacker aims at disturbing the devices' regular execution so that an error occurs. Based on a subsequent mathematical analysis of the genuine and the faulty response of the device, it might be possible to derive the used secret.

* The majority of his contribution was performed while he was with Ruhr-Universität Bochum.

An intuitive countermeasure to such attacks is to introduce redundancy by calculating twice, either in parallel (area redundancy or *duplication*) or consecutively (time redundancy) [9, 33]. When a fault is detected, the output is omitted or sensitive data is destroyed. However, only a single fault can be detected at the cost of at least twice the area or twice the computation time, respectively. Since the consistency of information is checked simultaneously with the computation, such schemes are usually denoted as Concurrent Error Detection (CED).

Recently, "higher-order" fault attacks have been demonstrated even in practice, i.e., targeting two independent spots with two laser beams [57] and creating multiple useful faults with only a single clock glitch [60]. Following the constant improvement of test equipment caused by advancing semiconductor technology, we expect to see even stronger adversaries in the future. To counteract, the scheme above can be extended easily to account for multiple faults, yet at a large overhead. Instead, multiple advanced techniques were proposed to improve fault detection capabilities or area/time penalties.

Error Detecting Codes (EDCs) seem to be a promising approach to counter strong adversaries as they can be easily adjusted by increasing the distance of the code, i.e., the maximum number of faults that the code can detect. However, the implementation of code-based CED suffers from certain problems: (a) the limited security of the sophisticated codes in practice, and (b) the higher complexity compared to plain duplication.

In theory, the security of a code-based CED is defined by the parameters of the employed code. In practice, however, it strongly depends on how the CED scheme is implemented. It is a trivial observation that one faulty gate can affect multiple subsequent gates. This effect which we later define as *fault propagation* can result in degradation of the achieved error-detecting capability compared to the one defined by the underlying code. While this is not considered an issue for duplication schemes, it can severely reduce the security of other more complex codes as we show later in this article. We present examples where a single faulty gate suffices to bypass advanced code-based CED schemes.

Contribution. In this paper, we present a methodology which enables the secure and practical implementation of code-based CED schemes in the presence of fault propagation. To this end, we first formally define the problem of *fault propagation* and highlight its consequences on the error-detection capability. Then, we present different strategies to limit its effect, each of which mitigates the security issue while having different area overheads. Consecutively, we define an adversary model, who is able to inject faults at a bounded number of cells at any location of the circuit (including data processing and control modules). On its basis, we present guidelines how to implement code-based CED in dedicated hardware circuits in such a way that the detection of faults fitting into the considered model is guaranteed. Note that opposed to considering only parity in [47], we provide generic and efficient strategies for arbitrary EDCs. We further cover every signal and component in our constructions including computational modules, finite state machine, and controlling signals. Indeed, it would not matter where the faults are injected, they must be detected as long as they are fitting to the considered bounded model. Similarly to Threshold Implementation (TI) [46] (which defines requirements to guarantee the security against side-channel analysis attacks up to a certain order), we define requirements to guarantee the security against fault attacks making use of up to certain number of faulty cells.

In order to explore the effectiveness of our methodology, we consider several case studies based on symmetric block ciphers including PRESENT [16], Skinny [12], Midori [7], GIFT [8], LED [32], and SIMON [11] with compatible state- and key sizes. We further cover different design architectures (round-based versus nibble-serial) as well as various linear EDCs with different distances to examine the area-overhead and throughput of our constructions by means of an ASIC standard cell library.

Related Work. There is an extensive body of work related to the design and implementation of CEDs. In some of such articles, the problem of fault propagation was already identified and some basic countermeasures were discussed. In the following, we briefly recall related works and indicate their limitations regarding the robust implementation of code-based CEDs in hardware.

Parity is often used as an EDC for CED schemes [13,45,47]. Within a Register Transfer Level (RTL) laser fault model, the authors of [5] presented a mechanism to reduce the area overhead by grouping the parity computation of independent sub-graphs. Furthermore, the authors of [47] identified the fault propagation issue and suggested dividing the circuit as a countermeasure, yet only within the context of parity-based schemes. In further previously-published articles [6, 18, 19], the use of other more sophisticated linear codes for CED schemes was proposed, including a formal verification of the error detecting capabilities. However, in most cases only a software implementation was considered, which limits its portability to hardware circuits. In [54], the authors explored how EDCs can be combined with TI [46] to construct an efficient hardware design resistant against both fault and side-channel attacks. They briefly referred to the effect of fault propagation and its danger to their construction. However, they did not formally provide dedicated solutions. Private Circuits II [34] is another approach aiming at designing a circuit protected against both active and passive adversaries. While it does provide provable security, the efficiency of its practical realization is questionable as shown in [22]. Recently, two new combined countermeasures based on multiparty computation have been proposed in [51, 55]. While the theoretical foundation of [55] seems sound, the paper does not include any implementations (apart from a SAGE implementation for simulations). Therefore, it is hard to assess the practical efficiency of the proposed scheme. For CAPA [51], the authors introduce a new formal adversary model including both active and passive attacks. Their approach is based on the concept of tiles and not affected by fault propagation, as they do not consider an adversary that is bounded by the number of faulty bits. Instead, they rely on the hardness of forging a valid MAC tag for fault resistance. Since it is a combined countermeasure (i.e., also providing side-channel resistance), it is not easily possible to compare their efficiency with our proposed method. With respect to setup time violation attacks, the problem of fault propagation was also discussed in [61]. The authors proposed a metric to evaluate the security of a given circuit called Timing Violation Vulnerability Factor (TVVF), which models the effect of fault propagation as a set of probabilities for each gate. It provides a good measure to compare the security of different circuits, but it is limited to a very specific type of attacks.

2 Preliminaries

2.1 Fault Injection Attacks

Boneh et al. [17] were the first who showed a practical fault attack by exploiting an erroneous computation on a cryptographic device to recover a secret key. For such attacks, the device is intentionally operated outside its specification so that some faulty output can be observed. Based on a subsequent mathematical analysis of the faulty (and genuine) output, the adversary is able to recover the secret key. Like all attacks that target the implementation of a cryptographic scheme, the success of a fault attack is mainly independent of the mathematical security considering a black-box model.

Physical means to inject a fault include tampering with the supply voltage [56] or the clock signal [2]. Both relate to a violation of the time the combinatorial logic requires for computation, either by slowing down the circuit (lower voltage) or by exceeding the maximum frequency (faster clock). Strong electromagnetic pulses [49] can affect the target's execution as well. While the aforementioned methods usually affect the whole device or a large area, optical fault injection [58] using focused laser beams can scale down the focus to a single transistor. Recent works have shown that with advanced optical setups it is even possible to target multiple transistors independently [57].

In practice, all physical fault injection techniques incorporate many parameters leading to vast search space for a successful attack. For example, the adversary must consider the timing (i.e., clock cycle), the physical intensity, and the duration of the effect. For targeted methods, the location (x/y) on the device and even the distance (EM) or focal plan (laser) is relevant as well. A useful fault might only occur if all parameters are correct. This already lead to various approaches trying to reduce the search space [20,53].

Multiple properties can be derived how the target will be affected. Most notably we can refer to its electrical effect, e.g., whether some internal value will be always set to logical '1' or always reset to logical '0'. Note that although faults sometimes are modeled as bit-flip or bit-toggle (i.e., set and reset based on the genuine value respectively) there is no reliable physical method known that would achieve this effect. In any case, bit toggle is certainly useful to model both set and reset faults. Another parameter is the area that will be affected, i.e., a single transistor, more bits, or the entire registers. A crucial aspect is the distribution of the resulting faults as there is usually some form of bias [2, 26, 28, 29, 31]. Considering for example clock glitches or underpowering, the bits involved in the critical path will be the first becoming faulty. For optical fault injections, only the exact area that is sufficiently illuminated will be affected.

From a mathematical point of view, the vast majority of attacks on ciphers is based on comparing a single or multiple faulty outputs to genuine ones respectively, so-called Differential Fault Analysis (DFA) [14]. However, there are multiple more "exotic" approaches that differ in certain aspects or requirements: Fault Sensitivity Analysis [44], Differential Fault Intensity Analysis [28], Statistical Fault Attacks [26], etc. Using one or another of such attacks, the implementations of both symmetric and asymmetric schemes were found to be vulnerable. In fact, nearly every newly-proposed cipher is usually followed by a publication describing a corresponding DFA, that is indeed a form of differential cryptanalysis on some last rounds of the cipher defined by a particular fault model.

Considering countermeasures, one approach is to shield the cryptographic operation in some way. This might include actual metal shields to hinder EM pulses, or generating the clock signals and voltage levels internally so that they cannot be affected by an attacker. Other countermeasures detect specific fault injection methods with particular sensors. However, such countermeasures are usually applied ad-hoc and counter only specific attacks.

An obvious generic approach is to introduce some form of redundancy in area and/or time. This translates to repeating the encryption and/or decryption for time redundancy, or multiple encryptions in parallel for area redundancy. More sophisticated approaches employ coding schemes instead of plain redundancy [5, 6, 13, 18, 19, 45, 47]. All CED schemes commonly check whether indeed no fault occurred to enable the output.

One interesting idea is Infective Computation which entirely omits the final check whether an error occurred. Instead, any faulty intermediate value will randomize the output of the cipher so that it is of no use for an attacker. Unfortunately, such approaches for symmetric schemes were repeatedly broken (cf. [10]).

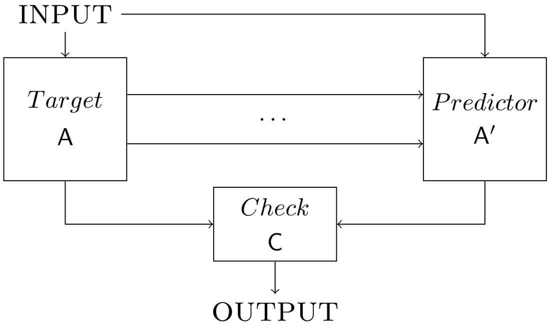


Fig. 1. Basic Structure of concurrent error detection schemes.

2.2 Concurrent Error Detection Schemes

As stated before, CED schemes are a common countermeasure against fault attacks. In the following, we briefly introduce the basic structure and corresponding notations, which we use in the rest of the paper.

Although a CED scheme can be realized in various ways with different types of redundancy, its basic structure is mostly the same. As depicted in Figure 1, these schemes usually include the original target algorithm A and its designated predictor A'. These predictors range from an exact duplicate of A in the most basic case (i.e., duplication) to sophisticated code-based predictors (e.g., parity). To increase the performance, some predictors are designed to operate on a compressed mapping of INPUT, e.g., only one bit for parity. Depending on A, such predictors may not be able to predict the compressed mapping of the output of A. Hence, they may require intermediate results from A during the computation. When both A and A' are finished, their results are checked in C to detect possible errors before transmitting OUTPUT. This structure is very generic and can be applied on different levels of granularity or types of redundancy.

Definition 1 (Fault Coverage). *The fault coverage of a given CED scheme* $\mathbf{C}$ *in a specific fault model* $\mathcal{M}$ *is defined as the ratio*

$$Cov_{\mathcal{M}}(\mathbf{C}) = \frac{\xi(\mathbf{C}, \mathcal{M})}{\psi(\mathbf{C}, \mathcal{M})},$$

where $\psi(\mathbf{C}, \mathcal{M})$ *(resp.* $\xi(\mathbf{C}, \mathcal{M})$*) stands for the number of possible (resp. detectable) faults of* $\mathbf{C}$ *adjusted to the distribution of* $\mathcal{M}$*.*

Such a metric to evaluate CED schemes has commonly been used in several related works, e.g., [13, 59]. While a higher fault coverage theoretically indicates a higher level of protection, the practical security strongly depends on the chosen fault model and its closeness to reality. This model should be carefully adapted based on the assumed adversary to avoid under- or overestimating the coverage. While underestimation reduces the security, overestimation results in inefficient implementations due to large overheads.

2.3 Error Detecting Codes

EDCs are an essential aspect of information theory and are often used in CED schemes to counter fault attacks. In the following, we introduce some notions [41] related to linear codes which are relevant to our work.

Definition 2 (Linear Code). *A binary linear* $[n, k]$*-code* $\mathbf{C}$ *is defined as a vector subspace over* $\mathbb{F}_2^n$ *which maps messages* $x \in \mathbb{F}_2^k$ *to codewords* $c \in \mathbf{C}$*.*

Most CEDs for symmetric cryptography rely on binary codes due to performance reasons and, thus, are the focus of our considerations. In the following, we refer to the code parameters n and k as the *length* and *rank* of an $[n, k]$-code $\mathbf{C}$.

Definition 3 (Generator Matrix). *A $k \times n$-matrix G is a generator matrix of an $[n, k]$-code $\mathbf{C}$ iff it consists of k basis vectors of $\mathbf{C}$ with length n. It can be used to map every message $x \in \mathbb{F}_2^k$ to its corresponding codeword with $x \cdot G = c \in \mathbf{C}$.*

Definition 4 (Minimum Distance). *The minimum distance d of a linear $[n, k, d]$-code $\mathbf{C}$ is defined as*

$$d = \min \left\{ wt\left(c_1 \oplus c_2\right) \mid c_1, c_2 \in \mathbf{C}, c_1 \neq c_2 \right\},$$

where $wt : \mathbb{F}_2^n \mapsto \mathbb{N}$ denotes the number of '1's in the binary representation, i.e, Hamming weight.

The error detection capability of a linear code $\mathbf{C}$ depends on its minimum distance, i.e., the larger the distance the more errors can be detected.

Lemma 1. *An $[n, k, d]$-code $\mathbf{C}$ can detect erroneous codewords $c' = c \oplus e$ iff $e \notin \mathbf{C}$.*

In particular, all error vectors $e \neq \mathbf{0}$ with $wt(e) \leq u = d - 1$ are detected.

Definition 5 (Systematic Code). *The generator matrix G of a systematic code $\mathbf{C}$ is of the form $G = [I_k|P]$ where I_k denotes the identity matrix of size k.*

Most CEDs utilize systematic codes, since they offer several beneficial implementation properties. Due to the structure of the generator matrix, each codeword c contains the unchanged message x which is padded by **check bits** x', i.e., $c = [x|p]$. The check bits can be easily generated using the matrix P as $x' = x \cdot P$. This special structure of the codewords enables a simple split of the data paths between message and check bits as depicted in Figure 1. Therefore, the original implementation of the target operation A can stay as it is, while it is extended with the predictors A' for the check bits. Furthermore, systematic codes do not require extra logic to recover the message from the codeword. As noted in [15], the focus on systematic linear codes does not pose a restriction, since every linear non-systematic code can be transformed into a systematic code with the same minimum distance.

Example 1 (Parity). Parity codes are a common approach to realize a basic and area-efficient CED scheme [5,13,45,47]. Since the check bits x' consists of only one additional bit, the required extra logic is rather small[3]. This leads to a $[k + 1, k, 2]$-code with an error-detecting capability of $u = 2 - 1 = 1$. Parity provides only full fault coverage for error vectors e with an odd weight.

Example 2 (Multiple Executions). Another common approach to CED schemes is to simply run the target algorithm multiple times (by either time or area redundancy) [33,37]. It turns to a $[\lambda k, k, \lambda]$-code where λ denotes the number of executions of the target algorithm (e.g., $\lambda = 2$ for running the target algorithm twice, i.e., duplication). The error-detecting capability $u = \lambda - 1$ can be straightforwardly improved by increasing λ at the cost of multiplying the overhead by λ.

Furthermore, in some articles it is proposed to use non-linear codes to improve the fault coverage [4,27,36,38,39]. However, their benefits over linear codes are questionable in some scenarios depending on the assumed fault model and overhead [42]. Nevertheless, many of the issues discussed in this paper are based on the linear property of the underlying code. Therefore, we particularly focus on binary linear codes in our constructions.

[3] Depending on the target algorithm the predictors can have an increased complexity, which diminishes the overhead advantage [42].

3 Concept

As stated before, many CED schemes rely on linear codes to provide fault detection. The effectiveness of this property heavily relies on the specific parameters of the underlying code, i.e., length, rank, and distance. While the length and rank directly affect the size of the code $|\mathbf{C}|$, the distance determines a lower bound for the Hamming weight of the error vector to which the code provides full fault coverage. In practice, these two parameters (size and distance) define the effective fault coverage of the implementation. However, many proposed CED schemes are either only theoretically evaluated in the common uniform fault model [13, 33] or by practical experiments which are limited by the capabilities of the evaluator [48]. In the uniform model, it is assumed that the error vector $e \in \mathbb{F}_2^n/\{\mathbf{0}\}$ follows a uniform distribution, i.e., $Pr(e) = \frac{1}{2^n-1}$, based on which the fault coverage is evaluated. Considering such an adversary model, the fault detection depends only on the code length and the rank.

Example 3 (Fault Coverage in the Uniform Model). The fault coverage of a CED scheme using an arbitrary $[n, k, d]$-code $\mathbf{C}$ can be easily computed in the uniform fault model (so-called $\mathcal{U}$). As noted in the previous section, an error vector e cannot be detected by the CED *iff* $e \in \mathbf{C}$. This relation allows us to trivially derive the fault coverage from the length n and rank k of the code as

$$Cov_{\mathcal{U}}(\mathbf{C}) = 1 - \frac{\mid \mathbb{F}_2^k/\{\mathbf{0}\} \mid}{\mid \mathbb{F}_2^n/\{\mathbf{0}\} \mid} = \frac{2^n - 2^k}{2^n - 1}. \tag{1}$$

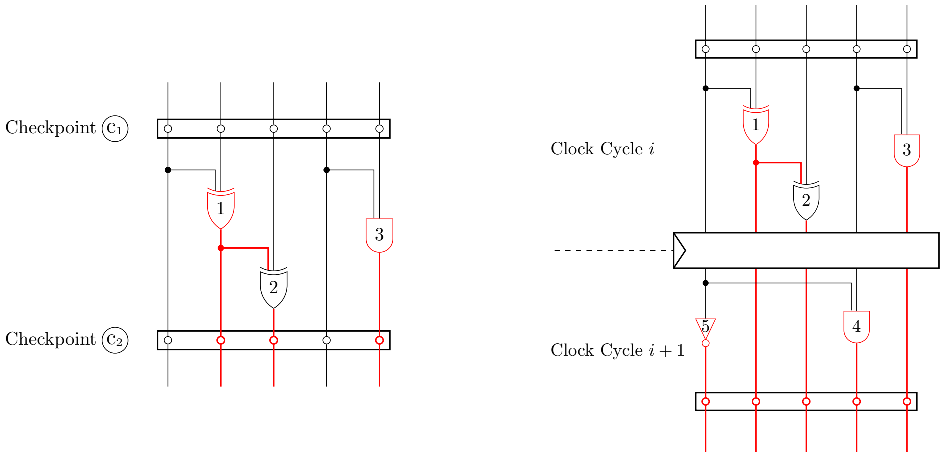
Since $Cov_{\mathcal{U}}$ is independent of the code distance d, every code with a constant length and rank provides the same fault coverage, e.g., an $[8, 4, 2]$-code $\mathbf{C}_1$ and an $[8, 4, 4]$-code $\mathbf{C}_2$ both have a fault coverage of $Cov_{\mathcal{U}}(\mathbf{C}_1) = Cov_{\mathcal{U}}(\mathbf{C}_2) = 0.94$.

However, the uniform fault model assumes a relatively weak adversary without much control over the fault injection process. In practice, most fault distributions, required by certain fault attacks, contain a specific bias as discussed in [2, 26, 28, 29, 31] which can increase the success rate of a potential attack. Furthermore, it is easy to see that the fault coverage can be drastically reduced if the fault distribution changes, e.g., attacking a duplication CED scheme by injecting a **symmetric fault**. Such a symmetric fault model requires that the same fault is injected into A and A'. These errors might be created using common injection techniques (e.g., clock glitching) leading to no detection for constructions with $\mathsf{A}' = \mathsf{A}$. In these scenarios, the distance of the code becomes an important factor for the effective fault coverage, e.g., when the adversary has highly accurate facilities to inject faults (e.g., laser fault injection targeting single bits [1, 3, 23, 52]). A higher distance increases the number of bits that need to be faulty to enable an attack.

The stronger an adversary is assumed to be, the more care needs to be taken when implementing a specific CED scheme. While EDC provides formal bounds on the error vectors, these bounds are only valid under certain assumptions. In the following, we first introduce our adversary model representing an attacker with varying control over the fault injection process. Then, we highlight the practical issue of **fault propagation** for code-based CED schemes and discuss critical design choices, which strongly improve the fault coverage of CED schemes in the presence of this issue. Some of these aspects have been identified in previous publications [5, 47, 54, 61]. We try to formalize them and provide a guideline how CED schemes should be integrated into the implementation of cryptographic algorithms.

3.1 Adversary Model

When considering CED schemes, the goal of the adversary is to create a faulty state that is not detected, leading to a faulty output that suits a DFA. We assume a similar



(a) univariate

(b) multivariate

Fig. 2. Fault injection by an $\mathcal{M}_{t=2}$ and $\mathcal{M}^*_{t=2}$ adversary.

adversary model to [34], i.e., the computation of the circuit is partitioned in clock cycles and the adversary can adaptively make t wires faulty (toggle) per clock cycle. If a wire is faulty, all its connections are also faulty, unless the attacker is able to cut a connection and make certain wire(s) faulty without affecting the other wires of the same connection. Each wire is the output of a cell (either a combinatorial gate or a register cell). Hence, we can model every fault on a wire as a fault on the corresponding cell.

Definition 6 (Adversary Model $\mathcal{M}_t$). *In a given sub-circuit, the adversary is able to make up to t cells (respectively wires) faulty at one clock cycle of the entire operation of the algorithm, e.g., a full encryption.*

Definition 7 (Adversary Model $\mathcal{M}^*_t$). *Here, the $\mathcal{M}_t$ adversary model is extended to allow the attacker to inject such bounded faults at multiple clock cycle.*

In other words, $\mathcal{M}_t$ is a **univariate** model while $\mathcal{M}^*_t$ refers to a **multivariate** scenario. In the following, we focus on $\mathcal{M}_t$ model and present techniques to construct a circuit providing full fault coverage. Afterwards, we give a solution to turn an $\mathcal{M}_t$-secure circuit to its $\mathcal{M}^*_t$-secure variant.

Definition 8 (Checkpoint). *A checkpoint* Ⓒ *monitors the correctness of the state of the circuit at a specific point in the computation.*

The check does not only include the data signals but extends to control signals as well. Furthermore, the frequency and position of the checkpoints inside the circuit strongly affect its security and efficiency. While too frequent checks will increase the area overhead and the critical path of the design, a missing checkpoint at an essential position could significantly reduce the fault coverage. Therefore, the module responsible for the consistency check is itself an attractive target for an adversary and needs to be implemented with care as we show later. Figure 2 depicts the concepts of checkpoints for an (a) univariate and (b) multivariate adversary with $t = 2$. While the univariate adversary can only make two cells faulty in one specific clock cycle (gates no. 1 and 3), the multivariate adversary is able to hit two cells per clock cycle with the result of making every output wire in the circuit faulty (gates no. 1 and 3 in the first clock cycle, and gates no. 4 and 5 in the second clock cycle).

The goal of EDC-based CED schemes is to set a relatively high lower bound for t, i.e., the attacker has to make **at least** t cells faulty to obtain a faulty output. This is based on the observation that the difficulty of fault attacks grows exponentially with increasing t, because of the increased parameter space (cf. Sect. 2.1). Note that we exclude Safe Error-like attacks (cf. [21, 40]). Such attacks are based on forcing a particular internal state to a known value, e.g., '0'. If no fault is observed, the adversary can conclude that the value was already '0'. This knowledge leads to very powerful attacks, e.g., when directly targeting the key register. Yet, as already discussed in [21, 40], despite its sophisticated and powerful adversary model, no algorithmic fault countermeasure without using any state randomization (e.g., Boolean masking) or error correction can thwart such attacks.

In this context, we want to note that while the paper focuses on error detection, the proposed methodology provides a sound basis to incorporate error correction in the circuit while still providing robustness again fault propagation. Since our approach already allows to correctly compute on encoded values, it would require only an additional correction module which needs to be implemented robustly. In this way, our methodology would achieve resilience against fault attacks immune to detection, e.g., statistical ineffective fault attacks (SIFA) [25], however, with a worse bound of $t = \lfloor \frac{d-1}{2} \rfloor$.

3.2 Fault Propagation

Below, we describe the observation that if an input of a gate in the circuit is faulty, its output might be faulty as well depending on the type of the gate and the value of the other inputs. This phenomenon is propagated through the circuit and as a result an $\mathcal{M}_t$-bounded adversary achieves $t + \delta \geq t$ faulty cells, where $\delta \geq 0$ depends on the t chosen cells, the underlying sub-circuit (i.e., the type of the cells forming the sub-circuit), and its input value. In the context of CED-based countermeasures, this aspect has been briefly considered in [5, 47, 54], and it is noted that it may reduce the fault coverage. We formally define fault propagation as follows.

Definition 9 (Fault Propagation). *We assume the worst case for fault propagation, i.e., one faulty input of a gate results in a faulty output. Hence, in the presence of fault propagation it is possible for an $\mathcal{M}_t$-bounded adversary to achieve t_p faulty gates with*

$$t \leq t_p \leq |\mathcal{G}|,$$

where $|\mathcal{G}|$ denotes the number of gates in the underlying sub-circuit.

This has serious implications for the security of code-based CED schemes as the distance of the underlying code does not provide a reliable bound for t anymore. In particular, this bound is only valid for the adversary who can only make faults at the output cells of the circuit, i.e., those cells whose output are exclusively connected to the checkpoints (i.e., no other fanout). Against an $\mathcal{M}_t$ adversary, however, who can arbitrarily make cells faulty, the distance of the code is only a rough indicator of security. We would rather have to design a CED scheme against an $\mathcal{M}_{t_p}$-bounded adversary to achieve security against any $\mathcal{M}_t$ adversary.

Example 4 (Parity with Fault Propagation). To illustrate the threat of fault propagation for CED schemes, we examine the susceptibility of a basic parity scheme. We assume a $[5, 4, 2]$-code $\mathbf{C}_{parity}$ with a generator matrix

$$G_{parity} = \begin{pmatrix} 1\,0\,0\,0\,1 \\ 0\,1\,0\,0\,1 \\ 0\,0\,1\,0\,1 \\ 0\,0\,0\,1\,1 \end{pmatrix} = \left(\begin{array}{c|c} 1\,0\,0\,0 & 1 \\ 0\,1\,0\,0 & 1 \\ 0\,0\,1\,0 & 1 \\ 0\,0\,0\,1 & 1 \end{array}\right) = (I \mid P_{parity}).$$

We also suppose a circuit realizing the function $T : \mathbb{F}_2^4 \mapsto \mathbb{F}_2^4$ which is supposed to be protected by such a CED. Exemplary, we consider a linear function $T(x) = x \cdot L$ with

$$L = \begin{pmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,1\,1\,0 \\ 0\,1\,1\,1 \end{pmatrix}.$$

Such a circuit protected by the $[5, 4, 2]$-code (including A and A') is depicted in Figure 3(a) consisting of three Exclusive OR (XOR) gates. A realizes $T(x) = x \cdot L$ and A' is formed by $x \cdot L \cdot P_{parity}$.

The checkpoints are also placed at the input and output of the circuit. In the uniform fault model, its fault coverage can be easily derived according to Equation (1) as

$$Cov_{\mathcal{U}}(\mathbf{C}_{parity}) = \frac{2^5 - 2^4}{2^5 - 1} = 0.52,$$

meaning that the countermeasure can detect around 52% of all injected faults. As stated before, the uniform fault model has a relatively low relevance in practice. Therefore, we compute the fault coverage under an $\mathcal{M}_t$ adversary model. Since the code has a distance of $d = 2$, we restrict the adversary to $t = d - 1 = 1$ gates. Without fault propagation, the code is indeed able to detect all possible faults injected at the gates whose output is exclusively connected to the checkpoints (e.g., XOR gates 2 and 3 in Figure 3(a)). Referring to such a fault model as $\mathcal{T}_{t=1}$, it results in

$$Cov_{\mathcal{T}_{t=1}}(\mathbf{C}_{parity}) = 1.$$

However, in the presence of fault propagation there are multiple possibilities that the adversary can increase the number of faulty gates and thus create a faulty state that is not detectable at the checkpoints. One of them is depicted in Figure 3(a) in which the adversary makes the XOR gate 1 faulty. It propagates through the XOR gate 2, and two faulty values arrive at the following checkpoint, creating a state that is not detectable by parity, i.e., the weight of the error vector is even. Repeating $\mathcal{M}_{t=1}$-bounded faults on all gates of the circuit results in

$$Cov_{\mathcal{M}_{t=1}}(\mathbf{C}_{parity}) = 2/3.$$

A commonly-believed simple solution to achieve the higher fault coverage would be to increase the distance of the code (similar to increasing the order in Boolean masking [43]). However, a code with a better error-detecting capability would probably require more area and results in a more complex circuit. These changes could amplify the fault propagation leading to a larger t_p possibly reducing the fault coverage even further.

Example 5 (Multiple Executions with Fault Propagation). An example for such type of redundancy is the realization of the target algorithm as multiple executions. Using the function $T(.)$ of our previous example, the code for duplication would have a generator matrix of

$$G_{dup} = \begin{pmatrix} 1\,0\,0\,0 & 1\,0\,0\,0 \\ 0\,1\,0\,0 & 0\,1\,0\,0 \\ 0\,0\,1\,0 & 0\,0\,1\,0 \\ 0\,0\,0\,1 & 0\,0\,0\,1 \end{pmatrix}.$$

The circuit is made up of two distinct realizations of A which do not share any intermediate wires. Under the uniform fault model, this scheme based on an $[8, 4, 2]$-code $\mathbf{C}_{dup}$ provides a fault coverage of

$$Cov_{\mathcal{U}}(\mathbf{C}_{dup}) = \frac{2^8 - 2^4}{2^8 - 1} = 0.94.$$

However, due to the particular structure of the circuit, fault propagation does not reduce the fault coverage considering an $\mathcal{M}_{t=1}$ adversary. Therefore, the coverage is

$$Cov_{\mathcal{T}_{t=1}}(\mathbf{C}_{dup}) = Cov_{\mathcal{M}_{t=1}}(\mathbf{C}_{dup}) = 1.$$

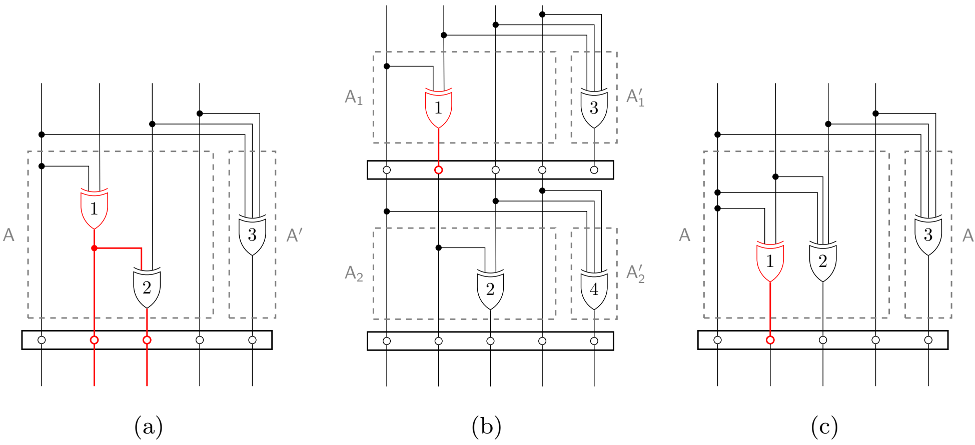


Fig. 3. Realization of T protected with the parity $[5, 4, 2]$-code. (a) undetectable fault with $t = 1$ faulty gates (red), (b) all $t = 1$ faulty gates detectable with an **extra checkpoint**, (c) all $t = 1$ faulty gates detectable with **forced independence**.

While duplication allows to trivially achieve the error-detection bound of the code, there are caveats to this approach in practice. For one, duplication comes with a considerable overhead. Secondly, due to the distance of the underlying code (i.e., $d = 2$) there is no guarantee to detect the error vectors e with $wt(e) > 1$. In particular, it is sufficient for an adversary to make the same gates in each duplicated circuit faulty to bypass the detection mechanism (i.e., symmetric fault). The fault coverage strongly depends on the target platform and the fault injection method. However, if we assume a symmetric fault model $\mathcal{S}_{2t}$, where up to t gates in each instance of the circuit can symmetrically be faulty, the fault coverage for duplication becomes

$$\forall t,\ Cov_{\mathcal{S}_{2t}}(\mathbf{C}_{dup}) = 0.$$

4 Methodology

In this section we present our solutions to provide full fault coverage under an $\mathcal{M}_t$ adversary model.

4.1 Extra Checkpoints

One strategy to restrict the negative impact of fault propagation is the inclusion of extra checkpoints in the circuit. The intuition behind this is very basic. If the design performs only a final check after the computation, all cells of the circuits can potentially contribute to the increased number of faulty cells. By splitting the circuit in smaller sub-circuits divided by checkpoints, this effect can be damped assuming each sub-circuit contains fewer gates than the whole design. Thereby, the effect of fault propagation is limited since a detectable faulty state cannot traverse over a checkpoint. This concept can be seen as the inclusion of registers in TIs of composed functions to prevent the propagation of glitches [46].

An interesting question is at which points in a circuit the extra checkpoints need to be inserted. An approach to reduce the effect of fault propagation for a given function $T(.)$ is a decomposition into multiple sub-functions as $T(x) = T_l \circ \ldots \circ T_1(x)$ with a checkpoint between every $T_i(.)$ and $T_{i+1}(.)$. This approach limits the fault propagation if each of the sub-functions is less sensitive to fault propagation. To measure the sensitivity to fault propagation, it can be trivially seen that the fault cannot be propagated if the circuit realizing a sub-function $T_i(.)$ has a depth of 1, i.e., there is no gate in the underlying sub-circuit whose input is derived from another gate of the same sub-circuit. Therefore, to completely remove fault propagation independent of the target function, it is necessary to include a checkpoint after every circuit depth in the combinatorial logic as noted in Lemma 2.

Lemma 2 (Preventing Fault Propagation with Checkpoints). *Fault propagation can be completely prevented by inserting a checkpoint at the output wires of all gates of a given circuit. To achieve this, the state at each checkpoint has to be a valid codeword under the employed code.*

Proof. The proof of Lemma 2 is straightforward. By checking every wire of every gate output, we prevent the propagation of one detectable faulty gate to undetectable multiple faulty gates. Therefore, one faulty gate can only result in maximum one faulty wire at the following checkpoint enforcing $t_p = t$. □

Example 6 (Parity with Extra Checkpoints). This strategy is very generic and can be applied independent of the target algorithm. It is only necessary to ensure that the state at every checkpoint is verifiable, i.e., a valid codeword. This can result in extra combinatorial logic as shown in Figure 3(b) for our previous parity example. The inclusion of an extra checkpoint in the middle of the circuit successfully prevents harmful fault propagation and ensures the error-detection capability for $t = 1$. To this end $T(x) = x \cdot L$ (see Section 3.2) is decomposed to $T = T_2 \circ T_1$ by

$$L_1 = \begin{pmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,1\,1 \end{pmatrix}, \qquad L_2 = \begin{pmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,1\,1\,0 \\ 0\,0\,0\,1 \end{pmatrix}.$$

Obviously, $\mathsf{A}_{i\in\{1,2\}}$ realizes $T_i(x) = x \cdot L_i$ and $\mathsf{A'}_i$ is formed by $x \cdot L_i \cdot P_{parity}$. Excluding the extra checkpoint, this comes at the cost of one additional XOR. Nevertheless, the new design $\widehat{\mathbf{C}}_{parity}$ provides the desired fault coverage of

$$Cov_{\mathcal{T}_{t=1}}(\widehat{\mathbf{C}}_{parity}) = Cov_{\mathcal{M}_{t=1}}(\widehat{\mathbf{C}}_{parity}) = 1.$$

4.2 Forced Independence

Another possibility to trivially achieve security against an $\mathcal{M}_t$-bounded adversary is based on the independence property of duplication. As noted in Section 3.2, the realization of the target function as λ instantiations of A results in the trivial bound of $t = d - 1 = \lambda - 1$, i.e., such a design (duplication/triplication/quadruplication/etc) provides security against an $\mathcal{M}_{t=\lambda-1}$-bounded adversary even with fault propagation.

Independence Property. The basic concept of independence can be generically applied to other codes as well. To this end, the circuit needs to be split up into independent component-circuits each computing exactly **one** output bit. Let us assume the target function $T : \mathbb{F}_2^k \mapsto \mathbb{F}_2^q$ which maps the input x to a q-bit output $y : \langle y^1, \ldots, y^q \rangle$. The function $T(x) = y$ is physically realized by q component-circuits each of which realizing a component-function $T^i : \mathbb{F}_2^k \mapsto \mathbb{F}_2^1$ in such a way that $\forall i,\ T^i(x) = y^i$. Such a set of component-circuits are call **independent** if no gate is shared between every two component-circuits.

$$\forall i, j;\ i \neq j \quad \mathcal{G}^i \cap \mathcal{G}^j = \varnothing,$$

where $\mathcal{G}^i$ stands for a set of gates implementing the component-function $T^i(.)$. The concept is formalized in Lemma 3.

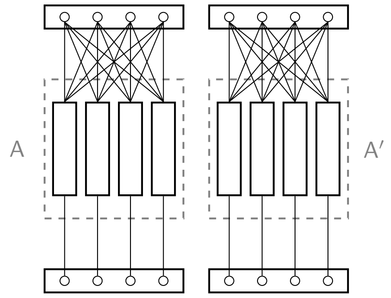


Fig. 4. Forced independence of the target algorithm A and its predictor A′.

Lemma 3 (Preventing Fault Propagation with Forced Independence). *Given a function $T : \mathbb{F}_2^k \mapsto \mathbb{F}_2^q$, a physical implementation realized by a set of q **independent** component-circuits does not suffer from fault propagation issue. Hence, $t_p = t$ is fulfilled if a checkpoint is placed at the output of $T(.)$.*

Proof. The proof of Lemma 3 is also trivial. Based on the assumption that the component-circuits (each of which realizing a component-function $T^i : \mathbb{F}_2^k \mapsto \mathbb{F}_2^1$) are distinct and do not share any resources, it is not possible for a faulty wire in T^i to traverse to $T^{j \neq i}$. Therefore, one faulty gate can maximally affect one output wire of $T(.)$, since each $T^i(.)$ computes only one unique output bit of $T(.)$. This trivially implies that every t faulty gates in the entire set of component-circuits can make at most t output wires of $T(.)$ faulty. Therefore, $t_p = t$. □

This strategy to thwart fault propagation is very simple to implement for a given function $T(.)$ by instantiating completely-independent component-circuits for every output bit (see Figure 4). Since the hardware synthesizers usually optimize the given design (e.g., by sharing the identical components to achieve lower area footprint), particular attention should be paid to avoid such optimizations[4]. Otherwise, the resulting circuit may merge the component-circuits which are supposed to be independent.

Example 7 (Parity with Forced Independence). For our running example, however, the strategy of forced independence can be readily applied. To this end, we consider A and A′ as one function $T : \mathbb{F}_2^5 \mapsto \mathbb{F}_2^5$. Based on Lemma 3, it is required to implement T as five distinct component-functions which do not share any resources. The resulting design $\bar{\mathbf{C}}_{parity}$ is depicted in Figure 3(c). Similar to the previous approach with an extra checkpoint, the design requires one more XOR gate. However, forced independence suffices with only one checkpoint which makes it more efficient than $\widehat{\mathbf{C}}_{parity}$, while providing the same fault coverage of

$$Cov_{\mathcal{T}_{t=1}}(\bar{\mathbf{C}}_{parity}) = Cov_{\mathcal{M}_{t=1}}(\bar{\mathbf{C}}_{parity}) = 1.$$

4.3 Combination

While each aforementioned solution independently can limit the negative effect of fault propagation, such designs are usually inefficient for complex cryptographic algorithms. Frequent checkpoints require additional combinatorial logic to ensure that the state is

[4] In common HDL designs, it can be done by instantiating a unique component for each component-function, and forcing the synthesizer to keep the hierarchy.

always a valid codeword. Depending on the underlying function, forced independence may require even more logic since ordinary optimizations (i.e., reuse of the common modules) become not allowed. Therefore, we propose to utilize a hybrid approach based on both strategies in three steps for a given target function $T(.)$.

1. *Decompose* $T : \mathbb{F}_2^k \mapsto \mathbb{F}_2^q$ *into multiple sub-functions* $T_i : \mathbb{F}_2^{k_i} \mapsto \mathbb{F}_2^{q_i}$ *of a less complexity while* $T = T_l \circ \ldots \circ T_1$, $k_1 = k$, $q_l = q$, *and* $k_{i+1} = q_{i<l}$.

Such a specific decomposition obviously depends on the target function $T(.)$. Considering a classical symmetric cipher, a trivial decomposition is to show every fundamental operation of the cipher (e.g., substitution, diffusion, and key addition) by a sub-function $T_i(.)$.

2. *Split each* $T_i(.)$ *into multiple smaller sub-functions* $T_{i,j} : \mathbb{F}_2^{k_{i,j}} \mapsto \mathbb{F}_2^{q_{i,j}}$ *with* $\sum_j k_{i,j} = k_i$ *and* $\sum_j q_{i,j} = q_i$.

In other words, each sub-function $T_i(.)$ is split into sub-functions with less input and output bits. A basic split in a majority of symmetric ciphers follows the cipher's structure where a certain function is applied multiple times in parallel, e.g., the S-box on each nibble or byte, and MixColumns on each word. In such cases, each sub-function $T_{i,j}$ would represent an S-box or a MixColumns. Obviously, step 1 and 2 can be repeated until the desired level of granularity is achieved.

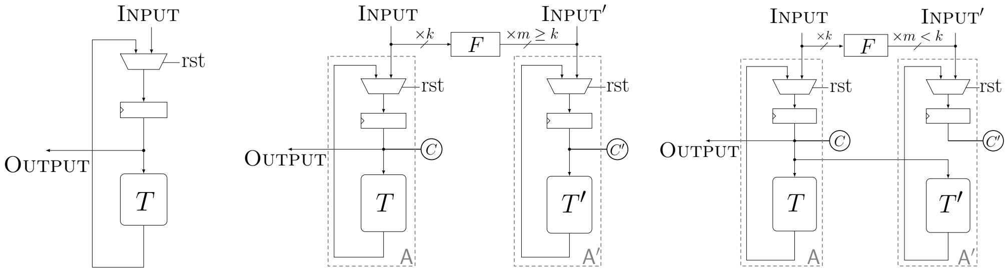
3. *Implement each sub-function* $T_{i,j}$ *fulfilling the independence property, and place a checkpoint at their output.*

This step benefits from the small input size of $T_{i,j}$, since it allows to reduce the area overhead immensely when the independence property should be complied. However, a decomposition (step 1) that is too fine would suffer from the basic problem of frequent checkpoints. Therefore, it is imperative to find a balance between the two strategies adjusted to the target function $T(.)$.

4.4 Application

In order to clarify the application of our strategies in a code-based CED scheme, let us consider an exemplary algorithm realized by the sequential circuit depicted in Figure 5(a) consisting of a register which loads the Input at the start of the operation (triggered by rst signal) and performs the function $T(.)$ repeatedly till the Output is taken from the register[5]. Note that any sequential circuit can be represented by such a construction. For the sake of simplicity let us suppose that the bit-length of Input, register, and $T(.)$ input and output is a multiple of k bits. The application of an $[n, k, d]$-code would lead to transforming every k-bit chunk x to an n-bit codeword $c = [x \mid x']$. Hereafter, we refer to the application of matrix P on x to derive the redundant part x' by $F : \mathbb{F}_2^k \mapsto \mathbb{F}_2^m$ as $F(x) = x \cdot P = x'$, where $m = n - k$ denotes the bit-length of the redundancy. Without losing the generality, we omit mentioning the bit-lengths of the message and the redundancy by a factor of k and m, respectively. Instead we use k and m (or $\times k$ and $\times m$) for simplicity. In the following we distinguish two different cases for m and explain how the underlying EDC is applied.

[5] Finite State Machine (FSM) is not shown.



(a) original (b) $m \geq k$ (c) $m < k$

Fig. 5. Our construction with respect to application of an EDC.

$\boldsymbol{m \geq k}$. If the redundancy size is at least as large as the message size, i.e., $n \geq 2k$, and the function $F(.)$ is injective[6], the redundancy part of the circuit (noted beforehand by A') can operate on x' independent of x, as shown in Figure 5(b). The redundant function $T'(.)$ is also trivially achieved as $T' = F \circ T \circ F^{-1}$, for any arbitrary function $T(.)$. Both $T(.)$ and $T'(.)$ are implemented following the forced independence lemma. Note that $T'(.)$ cannot be implemented as separate decomposed functions $F^{-1}(.)$, $T(.)$, and $F(.)$. Instead, its description should be first derived (as given above) and then implemented by independent component-functions. As the last step, a single checkpoint is placed at the input of the $T(.)$ function (marked by Ⓒ and Ⓒ′ in Figure 5(b)). Applying the hybrid combination technique (given in Section 4.3) on $T(.)$ is straightforward leading to more checkpoints between every two consecutive decompositions (see Figure 6(a)).

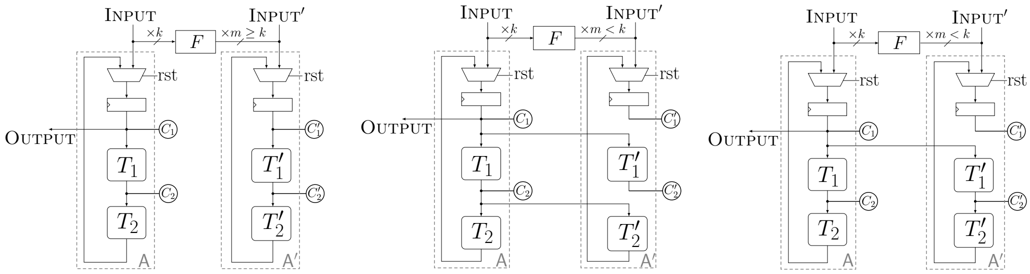
An arising question is whether it is essential to place the checkpoint at the input of $T(.)$. Otherwise, if the checkpoints is moved to the output of $T(.)$, the faults injected at the register cells would potentially propagate to multiple output bits of $T(.)$. Hence, in order to maintain full fault coverage against an $\mathcal{M}_{t=d-1}$-bounded adversary, the checkpoint should be placed right at the input of the function implemented by the forced independence lemma. It should be noted that since the multiplexer and the register (see Figure 5(b)) independently operate on each bit of the $T(.)$ output, they do not affect the independent property of the entire circuit (i.e., from the $T(.)$ input forward to the register output). Therefore, any fault injected at $T(.)$ fitting to $\mathcal{M}_{t=d-1}$, is detected at the checkpoint in the next clock cycle.

$\boldsymbol{m < k}$. For the cases, where smaller redundancy $n < 2k$ is desired (for lower area overheads), the function $F(.)$ cannot obviously be injective. Hence, the construction shown in Figure 5(b) is not necessarily applicable. It indeed depends on the specification of the underlying function $T(.)$. For an arbitrary $T(.)$, it is not always possible for $T'(.)$ to solely operate on x'. In such cases, $T'(.)$ needs to receive the original data x instead to be able to compute $T'(x) = F \circ T(x)$. The corresponding construction is shown in Figure 5(c), where the only difference to the former case ($m \geq k$) is how the $T'(.)$ is realized. It is noteworthy that – similar to before – the implementation of $T'(.)$ should also fulfill the independence property.

We have observed that for some particular functions $T(.)$ it is still possible to realize the circuit similar to the one shown in Figure 5(b), i.e., $T'(.)$ can operate only on x' while $m < k$. Any intermediate value of the circuit should be a valid codeword. In other words, if $x \in \mathbb{F}_2^k$ and $x' \in \mathbb{F}_2^m$ are the input of $T(.)$ and $T'(.)$ respectively with $x' = F(x)$, their output $\langle T\left(x\right), T'\left(x'\right)\rangle$ should also form a valid codeword. This implies that $T'(x') = F\left(T\left(x\right)\right)$, i.e.,

$$T' \circ F = F \circ T. \tag{2}$$

[6] Note that if $F(.)$ is non-injective, then it can be considered as a similar case to $m < k$.



(a) $m \geq k$ (b) $m < k$ (c) $m < k$

Fig. 6. Our construction with respect to application of an EDC with decomposition.

Given a $T(.)$ and $F(.)$, it can be examined if there exists such a function $T'(.)$ fulfilling the above condition. In many cases, specially in SPN block ciphers, the linear layer uses multiplication in $\mathbb{F}_{2^k}$, i.e., $T : \mathbb{F}_{2^k} \mapsto \mathbb{F}_{2^k}$ in such a way that $T(x) = a \bullet x$ with constant $a \in \mathbb{F}_{2^k}$. In the following we show that if $a \neq \{0, 1\}$ then there exists no such a function $T'(.)$ fitting to Equation (2). Note that for simplicity we used $F : \mathbb{F}_{2^k} \mapsto \mathbb{F}_{2^m}$ instead of $F : \mathbb{F}_2^k \mapsto \mathbb{F}_2^m$ (there is always a bijective mapping between $\mathbb{F}_2^k$ and $\mathbb{F}_{2^k}$).

Lemma 4. *Let $T : \mathbb{F}_{2^k} \mapsto \mathbb{F}_{2^k}$ that $T(x) = a \bullet x$ with $a \in \mathbb{F}_{2^k}/\{0, 1\}$, and let $F : \mathbb{F}_{2^k} \mapsto \mathbb{F}_{2^m}$ be any linear function with $m < k$. Then, there is no $T' : \mathbb{F}_{2^m} \mapsto \mathbb{F}_{2^m}$ such that $F \circ T = T' \circ F$.*

Proof. Since $F(.)$ is a linear function, among its inputs there exist 2^c (with $c \geq k - m$) values which are mapped to zero. In other meaning, there are $2^c - 1 \geq 1$ nonzero roots for $F(.)$. So, let u be one of these nonzero roots, i.e., $F(u \neq 0) = 0$. Now assume that there exists a $T'(.)$ function which $F \circ T = T' \circ F$. It is clear that as both $F(.)$ and $T(.)$ are linear, $T'(.)$ is also a linear function. This results to $T'(0) = 0$. Hence, we have

$$F \circ T(u) = T' \circ F(u) = T'(0) = 0 \quad \Rightarrow \quad F(a \bullet u) = 0,$$

which means $a \bullet u$ is another nonzero root of $F(.)$. By repeating above equation, we find out that for any $i \geq 0$, $a^i \bullet u$ is a nonzero root for $F(.)$. Since $T(.)$ is a multiplication in $\mathbb{F}_{2^k}$, none of $(a^i \bullet u, a^j \bullet u)$ with $i \neq j$ and $i, j < 2^k - 1$ are equal to each other. Hence, we have $2^k - 1$ nonzero roots for $F(.)$ which means that any $x \in \mathbb{F}_{2^k}$ is a root for $F(.)$ (i.e., $\forall x, F(x) = 0$) that is in contrast with our assumption that $F(.)$ is an arbitrary linear function. □

Although with $a \neq \{0, 1\}$ there is no solution for Equation (2), it has obvious solutions for $a = 0$ or $a = 1$. It means that, if $T(.)$ is a multiplication with $a = 0/1$ in $\mathbb{F}_{2^k}$, the redundant counterpart $T'(.)$ is a multiplication with the same constant in $\mathbb{F}_{2^m}$. The decomposition of $T = T_2 \circ T_1$ would generally lead to a construction similar to Figure 6(b), where a checkpoint is placed between the composed sub-functions implemented complying the independence property. However, if the linear layer of a block cipher utilizes multiplications by only zero or one, then it is possible to realize the EDC-equipped circuit with the architecture shown in Figure 6(c), which is the case for several block ciphers including Midori [7], Skinny and Mantis [12].

Optimization. Suppose that $T(.)$ is decomposed as $T = T_2 \circ T_1$. Further, suppose that – independent of the redundancy size m – the second sub-function $T'_2(.)$ can solely operate on the output of $T'_1(.)$, i.e., either Figure 6(a) or Figure 6(c). In the following, we present an observation that for certain *linear* sub-functions $T_2(.)$, it is not necessary to place a checkpoint between the composed sub-function $T_2 \circ T_1$, i.e., the checkpoint marked by $\textcircled{C_2}$ and $\textcircled{C'_2}$ in Figure 6(a) and Figure 6(c).

Theorem 1. *Let us represent the linear function $T_2(.)$ by matrix L with elements in $\mathbb{F}_{2^k}$. The extra check* $\textcircled{c_2}$ *and* $\textcircled{c_2'}$ *between $T_1(.)$ and $T_2(.)$ is not required if L is formed by only 0 or 1.*

Proof. Below, we represent an intermediate value of the circuit by $\boldsymbol{x}$ (resp. $\boldsymbol{x}'$) as s equally-sized k-bit chunks $\langle x_1, \ldots, x_s \rangle$ (resp. m-bit chunks $\langle x'_1, \ldots, x'_s \rangle$). Due to the independence property of the implementation of each sub-function, any fault injected at t cells of a sub-function can be modeled by an additive error vector $\boldsymbol{e}$ at its output. We use the notation $\boldsymbol{e}_1 = \langle e_{1,1}, \ldots, e_{1,s} \rangle$, $\boldsymbol{e}'_1 = \left\langle e'_{1,1}, \ldots, e'_{1,s} \right\rangle$, $\boldsymbol{e}_2 = \langle e_{2,1}, \ldots, e_{2,s} \rangle$ and $\boldsymbol{e}'_2 = \left\langle e'_{2,1}, \ldots, e'_{2,s} \right\rangle$ for the corresponding error vectors of injected faults in $T_1(.)$, $T'_1(.)$, $T_2(.)$ and $T'_2(.)$, respectively. The check at $\textcircled{c_1}$ and $\textcircled{c_1'}$ examines indeed the equality of the below equation:

$$F(T_2(\boldsymbol{x} \oplus \boldsymbol{e}_1) \oplus \boldsymbol{e}_2) \overset{?}{=} T'_2(\boldsymbol{x}' \oplus \boldsymbol{e}'_1) \oplus \boldsymbol{e}'_2,$$

where $\boldsymbol{x}$ denotes the fault-free output of $T_1(.)$, i.e., the input of $T_2(.)$. We already know that $\boldsymbol{x}' = F(\boldsymbol{x})$ and $F \circ T_2 = T'_2 \circ F$, which simplifies the above equation to

$$F(T_2(\boldsymbol{x})) \oplus F(T_2(\boldsymbol{e}_1)) \oplus F(\boldsymbol{e}_2) \overset{?}{=} T'_2(\boldsymbol{x}') \oplus T'_2(\boldsymbol{e}'_1) \oplus \boldsymbol{e}'_2 \quad \Rightarrow$$

$$F(T_2(\boldsymbol{e}_1) \oplus \boldsymbol{e}_2) \overset{?}{=} T'_2(\boldsymbol{e}'_1) \oplus \boldsymbol{e}'_2 \tag{3}$$

Considering an $\mathcal{M}_{t=d-1}$ adversary model, suppose that the attacker injects such bounded faults, i.e., $wt(\boldsymbol{e}_1) + wt(\boldsymbol{e}'_1) + wt(\boldsymbol{e}_2) + wt(\boldsymbol{e}'_2) < d$. In order to detect such a fault at checkpoint $\textcircled{c_1}$ and $\textcircled{c_1'}$, the relation in Equation (3) should be unequal. In other meaning

$$\forall \boldsymbol{e}_1, \boldsymbol{e}'_1, \boldsymbol{e}_2, \boldsymbol{e}'_2\,; 0 < wt(\boldsymbol{e}_1) + wt(\boldsymbol{e}'_1) + wt(\boldsymbol{e}_2) + wt(\boldsymbol{e}'_2) < d \implies F(T_2(\boldsymbol{e}_1) \oplus \boldsymbol{e}_2) \neq T'_2(\boldsymbol{e}'_1) \oplus \boldsymbol{e}'_2$$

which equally means

$$\begin{aligned} &\forall \boldsymbol{e}_1, \boldsymbol{e}'_1, \boldsymbol{e}_2, \boldsymbol{e}'_2\,; F(T_2(\boldsymbol{e}_1) \oplus \boldsymbol{e}_2) = T'_2(\boldsymbol{e}'_1) \oplus \boldsymbol{e}'_2 \implies \\ &\quad wt(\boldsymbol{e}_1) + wt(\boldsymbol{e}'_1) + wt(\boldsymbol{e}_2) + wt(\boldsymbol{e}'_2) \geq d \ \vee \ wt(\boldsymbol{e}_1) = wt(\boldsymbol{e}'_1) = wt(\boldsymbol{e}_2) = wt(\boldsymbol{e}'_2) = 0 \end{aligned} \tag{4}$$

With respect to the linear property of $T_2(.)$ and $T'_2(.)$, we define L and L' matrices as follows

$$T_2(\boldsymbol{x}) = \langle x_1, \cdots, x_s \rangle \cdot L \quad , \quad T'_2(\boldsymbol{x}) = \langle x'_1, \cdots, x'_s \rangle \cdot L'$$

$$L = \begin{pmatrix} L_{11} & L_{12} & \cdots & L_{1s} \\ L_{21} & L_{22} & \cdots & L_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ L_{s1} & L_{s2} & \cdots & L_{ss} \end{pmatrix} \quad , \quad L' = \begin{pmatrix} L'_{11} & L'_{12} & \cdots & L'_{1s} \\ L'_{21} & L'_{22} & \cdots & L'_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ L'_{s1} & L'_{s2} & \cdots & L'_{ss} \end{pmatrix},$$

where each L_{ij} and L'_{ij} are binary $k \times k$ and $m \times m$ matrices, respectively, and

$$\forall i, j \quad L_{ij} \cdot P = P \cdot L'_{ij}.$$

Note that P is the part of the generator matrix of the underlying code in such a way that $F(\boldsymbol{x}) = \langle x_1 \cdot P, \ldots, x_s \cdot P \rangle$. Using above definitions, $F(T_2(\boldsymbol{e}_1) \oplus \boldsymbol{e}_2) = T'_2(\boldsymbol{e}'_1) \oplus \boldsymbol{e}'_2$ can be written as following s equations

$$\begin{aligned} \Big(\bigoplus_{j=1}^{s} e_{1j} \cdot L_{j1} \oplus e_{21} \Big) \cdot P \quad &= \quad \bigoplus_{j=1}^{s} e'_{1j} \cdot L'_{j1} \oplus e'_{21} \\ &\vdots \\ \Big(\bigoplus_{j=1}^{s} e_{1j} \cdot L_{js} \oplus e_{2s} \Big) \cdot P \quad &= \quad \bigoplus_{j=1}^{s} e'_{1j} \cdot L'_{js} \oplus e'_{2s} \end{aligned}$$

Let us denote $\alpha_i = \bigoplus_{j=1}^{s} e_{1j} \cdot L_{ji} \oplus e_{2i}$ and $\beta_i = \bigoplus_{j=1}^{s} e'_{1j} \cdot L'_{ji} \oplus e'_{2i}$. If $\forall i\ \alpha_i = 0$, then $T_2(\boldsymbol{e}_1) \oplus \boldsymbol{e}_2 = 0$. It means that the output of $T_2(.)$ is fault free, hence not useful for the adversary. Therefore, we can conclude that

$$\forall \boldsymbol{e}_1, \boldsymbol{e}'_1, \boldsymbol{e}_2, \boldsymbol{e}'_2\ ; F(T_2(\boldsymbol{e}_1) \oplus \boldsymbol{e}_2) = T'_2(\boldsymbol{e}'_1) \oplus \boldsymbol{e}'_2 \implies \boldsymbol{e}_2 = T_2(\boldsymbol{e}_1)\ \vee\ \exists i;\ \alpha_i \neq 0.$$

So without loss of generality, we consider that there exists an i which $\alpha_i \neq 0$ and $\beta_i = \alpha_i \cdot P$. For an $[n, k, d]$-code, we already know that for any nonzero x, $wt(x) + wt(x \cdot P) \geq d$ which implies that

$$\alpha_i \neq 0 \implies wt(\alpha_i) + wt(\alpha_i \cdot P) = wt(\alpha_i) + wt(\beta_i) \geq d. \tag{5}$$

As stated, every L_{ji}/L'_{ji} is either zero or identity matrix. Hence, $e_{1j} \cdot L_{ji}$ can be considered as a scalar product of $e_{1j} \cdot a_{ji}$ with $a_{ji} = 0/1$. Therefore, we can simplify Equation (5) as follows.

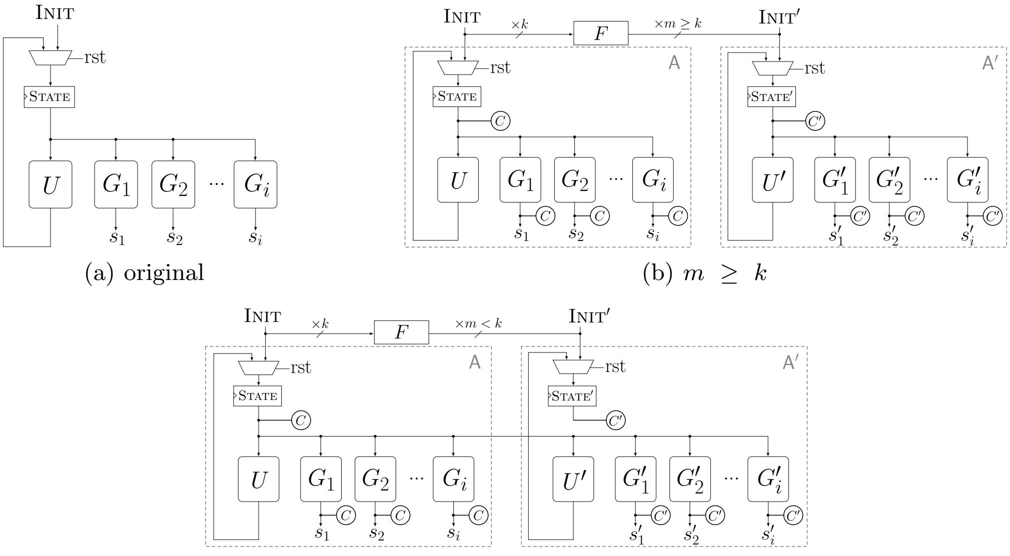
$$\begin{aligned}
\alpha_i \neq 0 \implies d \leq wt(\alpha_i) + wt(\beta_i) &= wt\Big(\bigoplus_{j=1}^{s} e_{1j} \cdot a_{ji} \oplus e_{2i}\Big) + wt\Big(\bigoplus_{j=1}^{s} e'_{1j} \cdot a_{ji} \oplus e'_{2i}\Big) \\
&\leq wt\Big(\bigoplus_{j=1}^{s} e_{1j} \cdot a_{ji}\Big) + wt\Big(\bigoplus_{j=1}^{s} e'_{1j} \cdot a_{ji}\Big) + wt(e_{2i}) + wt(e'_{2i}) \\
&\leq \sum_{j=1}^{s} wt(e_{1j} \cdot a_{ji}) + \sum_{j=1}^{s} wt(e'_{1j} \cdot a_{ji}) + wt(e_{2i}) + wt(e'_{2i}) \\
&\leq \sum_{j=1}^{s} wt(e_{1j}) + \sum_{j=1}^{s} wt(e'_{1j}) + wt(e_{2i}) + wt(e'_{2i}) \\
&= wt(\boldsymbol{e}_1) + wt(\boldsymbol{e}'_1) + wt(e_{2i}) + wt(e'_{2i}) \\
&\leq wt(\boldsymbol{e}_1) + wt(\boldsymbol{e}'_1) + wt(\boldsymbol{e}_2) + wt(\boldsymbol{e}'_2)
\end{aligned}$$

□

We have observed that the corresponding matrix of MixColumns in Skinny [12] and Midori [7] fulfill this condition, enabling to avoid the extra checkpoint.

Control Signals. In the examples shown above we excluded the Finite State Machine (FSM) in our discussions. In contrast to masking countermeasures (against side-channel analysis attacks), the FSM should be also protected against faults. Otherwise, the adversary can change the control flow and obtains faulty results exploiting the secrets. As a trivial example, independent of the employed EDC scheme on the data-processing part of the circuit, the attacker can force to terminate an encryption process at the first cipher rounds. This leads to having access to the cipher intermediate values, hence recovering the key.

The FSM can also be seen as a set of register cells loaded by a certain Init value, and updated at every clock cycle through an update function $U(.)$. We refer to the content of the register by State. Each control signal s_i is derived by a dedicated function over the FSM register, marked by $G_i(.)$ in Figure 7(a). The application of an $[n, k, d]$-code on such a controlling circuit is the same as what shown above. The only difference is how the control signals are encoded. Each control signal s_i and its redundant counterpart s'_i are related in a form of $s'_i = F(\{0\}^{k-1}|s_i)$, i.e., s_i is padded with zero to form a k-bit chunk. In other words, the redundancy of every single-bit control signal has a size of m bits.



(a) original

(b) $m \geq k$

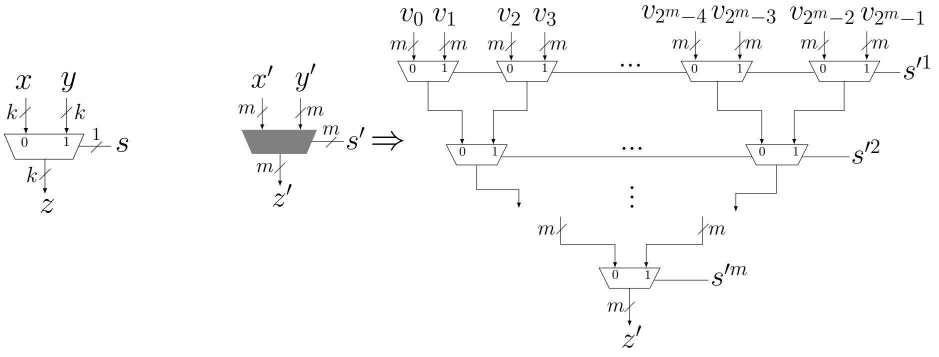
(c) $m < k$

Fig. 7. Our construction with respect to application of an EDC on FSM.

- For $m \geq k$, the redundant part of the update function would realize $U' = F \circ U \circ F^{-1}$ over State′ (i.e., the redundant part of State). Each control signal s_i is mapped to $s'_i = G'_i(\text{State}')$ with $G'_i = F \circ G_i \circ F^{-1}$. Figure 7(b) shows an exemplary construction.
- For $m < k$, the redundancy update function would operate on State as $U' = F \circ U$. The same holds for the control signals as $s'_i = G'_i(\text{State})$ while $G'_i = F \circ G_i$ (see Figure 7(c)). Note that, depending on $G_i(.)$ it might be possible to generate s'_i over State′. To this end, there should exist a $G'_i(.)$ satisfying $G'_i \circ F = F \circ G_i$.

It is important to emphasize that all above-given functions $U(.)$, $U'(.)$, $G_i(.)$, and $G'_i(.)$ should be implemented following the forced independence lemma. As shown in Figure 7, the checkpoints are placed at the register output as well as at the output of every function generating a control signal s_i. As a side note, it is possible to merge a couple of control signals before encoding them, but it usually leads to a more complicated multiplexer which is controlled by such a redundant control signal. We give details about redundant multiplexers in the following.

Multiplexers. As stated before, each redundant control singal s' is an m-bit word. This is essential since otherwise if the redundant counterpart of each control signal is also single-bit wide, similar to the duplication scheme the control signals become vulnerable to symmetric faults independent of the distance of the underlying EDC. Suppose that the signal s controls a k-bit multiplexer switching between x and y (see Figure 8(a)). The redundant counterpart should be a multiplexer switching between m-bit x' and y' words by an m-bit redundant control signal s'. To this end, we propose the construction shown in Figure 8(b) formed by a multiplexer tree in m levels. Each row of the multiplexers is controlled by the corresponding bit of the redundant control signal. The first row by the Least Significant Bit (LSB), i.e., s'^1, and the last row by MSB s'^m. The m-bit inputs



(a) original (b) redundancy

Fig. 8. Our construction with respect to application of an EDC on multiplexers.

$v_{i\in\{0,\ldots,2^m-1\}}$ of the first multiplexer row are defined as follows:

$$v_i = \begin{cases} x' \;; & i = F(0) \\ y' \;; & i = F(1) \\ 0 \;; & else^7 \end{cases}$$

$\left\langle \{0\}^{k-1} \mid s, s' \right\rangle$ forms two n-bit valid codewords, hence with minimum distance d. Therefore, considering the fact that the component-functions generating the control signals and their redundant counterparts (i.e., $G_i(.)$ and $G'_i(.)$) fulfill the independence property, this construction guarantees the detection of $t < d$ faulty gates at the control logic.

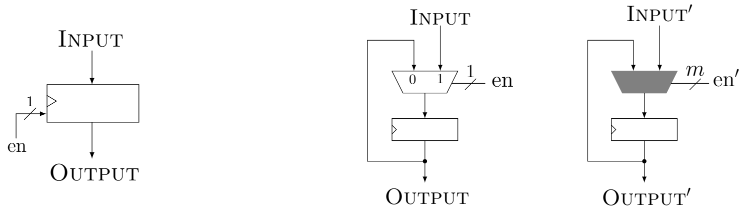
Note that since all input signals v_i except two are connected to zero, the synthesizer usually optimizes this construction and removes those 2-to-1 multiplexers whose both inputs are tied to zero independent of the select signal. Such an optimization does not affect the fault propagation and hence the fault coverage of our construction.

We would also like to highlight that the faults on external signals, those provided through the I/O ports of the circuit, cannot be internally detected. For instance, any fault on plaintext of an encryption function is interpreted as encrypting another plaintext and does not lead to any exploitable information about the secrets involved in the encryption function. Therefore, the external control signals (e.g., the reset signal marked by rst in Figure 5 and Figure 7) as well as the multiplexers which are controlled by such external signals do not have to be encoded. In other words, the same external signal is used in both original A and redundant part A' of the circuit. This can be seen in Figure 5, Figure 6 and Figure 7.

Registers with Enable. If the underlying circuit contains registers with enable signal, the redundant counterpart cannot trivially make use of the corresponding redundant control signal with bit-length $m > 1$. Therefore, we propose the solution shown in Figure 9 to replace such registers with their equivalent construction formed by a register without enable and a multiplexer. This makes it enable to employ the above-explained redundant multiplexer controlled by redundant control signal.

Checkpoints. All the above given structures do not cover how the consistency check at the checkpoints is performed. It is of great importance to integrate a fault detection mechanism into the consistency check process as well. Otherwise, the attacker can target

[7] Arbitrary random values can be given to those inputs v_i which are tied to zero without affecting the fault coverage.



(a) original

(b) transformed

Fig. 9. Our construction with respect to application of an EDC on registers with enable.

the final module and force a faulty output to pass the consistency check process, independent of the fault coverage of the data-processing part of the circuit. Therefore, it is necessary to be able to detect up to $t = d - 1$ faults at the consistency check process in order to provide full fault coverage on the entire circuit against an $\mathcal{M}_t$-bounded adversary. To cope with this issue, we propose the construction shown in Figure 10. The values of the checkpoints at the original part of the circuit $\textcircled{c}_i$ are concatenated[8] and each k-bit chunk is given to an instance of $F(.)$ function, whose output is marked by c''. Its consistency is examined with the value of all checkpoints at the redundant part concatenated together, marked by c'. Un-ordinarily the result of such a consistency check is an m-bit error vector e. To this end, c'' and c' are split into m chunks[9]. The i-th bit of the error vector examines the consistency of all bits of the corresponding i-th chunks:

$$e^{i\in\{1,\ldots,m\}} \, : \, \left\langle c''^{i}, c''^{i+m}, c''^{i+2m}, \ldots \right\rangle \overset{?}{=} \left\langle c'^{i}, c'^{i+m}, c'^{i+2m}, \ldots \right\rangle .$$

As shown in Figure 10, all bits of the error vector are ORed with the entire $d-1$ bits of the error register $\widehat{e}$, before being stored in the same register. Such a register is reset by the rst signal, the same signal which starts the operation of the circuit and the FSM. This construction implies that once an error is detected, the full content of the error register is filled by '1' and stays unchanged till the next reset phase. As the last step, the $d-1$ bits result of the OR operation (marked by $\widetilde{e}$) controls a redundant multiplexer with $d-1$ bits control signal (see Figure 8(b)). Such a multiplexer should pass the Output when all $d-1$ bits of the control signal $\widetilde{e}$ are zero. Therefore, with respect to the construction shown in Figure 8(b), the input signals of the multiplexer are selected as follows:

$$v_i = \begin{cases} \text{Output} \; ; & i = 0 \\ 0 \qquad\quad\; ; & else \end{cases}$$

Finally, the output of the multiplexer is stored in a dedicated register providing the final output of the circuit. The register is reset by the rst signal and stores the multiplexer output when the computation of the circuit is finished. This is identified by the done signal, which is amongst the control signals derived from the State of the FSM, and its consistency is also examined similar to other control signals. By detecting even a single-bit fault, the entire $d-1$ bits vector $\widetilde{e}$ becomes '1'. It should be noted that although all bits of $\widetilde{e}$ are the same, the independence property should be also fulfilled in the implementation of the OR operation realizing each bit of $\widetilde{e}$. As a simple example, suppose that the attacker made a single cell faulty (in any part of the circuit). This causes $\widetilde{e}$ to be fully '1'. In order to force the faulty Output pass the multiplexer, this implies that the attacker needs to make at least $d-1$ more cells faulty at the same clock cycle (due to the underlying univariate model) to turn $d-1$ bits vector $\widetilde{e}$ with value $\{1\}^d$ to $\{0\}^d$ This achieves full fault coverage on the entire circuit under the $\mathcal{M}_{t=d-1}$-bounded adversary.

[8] As stated before, the single-bit control signals are padded with zero before being concatenated with others.

[9] The size of both c' and c'' is always a factor of m bits.

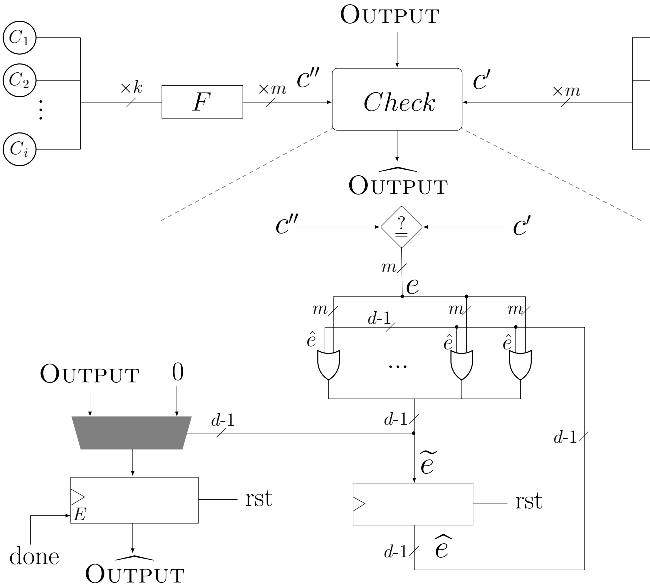


Fig. 10. Our construction with respect to application of an EDC on the consistency check.

4.5 Extension to Multivariate

As defined in Section 3.1, under the $\mathcal{M}_t$ model the adversary is able to make at most t cells faulty at one clock cycle of the entire operation of the algorithm, i.e., between two consecutive reset phases. Suppose the circuit shown in Figure 5(b) with an $[n, k, d]$-code and $d > 1$, which should detect all single-cell faults. Suppose also that at one clock cycle before the last, the adversary makes a gate in $T(.)$ faulty, which results in a value with a single-bit fault stored in the register. If at the next clock (which is the last one) the adversary injects a single-gate fault on the corresponding $F(.)$ function of the consistency check process (Figure 10), the faulty output can pass the multiplexer and be stored in the final register. In order to extend the adversary model to multivariate in order to be able to keep the full fault coverage even if the adversary makes up to t-cell faults at every clock cycle, it suffices to just introduce extra checkpoints right at the input of the registers in the design. Independent of the redundancy size m, this includes the register of the data-processing module and that of the FSM (see Figure 11). This makes sure that the consistency of the values stored in the registers is examined. Hence, in the aforementioned example the fault is detected at first clock cycle. Introducing more checkpoints obviously increases the area requirements, further since they are placed right at the input of the registers, the critical path delay of the circuit is increased leading to lower throughput.

We further need to adjust the consistency check process to support the multivariate $\mathcal{M}_t^*$ model. To this end we have to increase the size of the error register $\widehat{e}$ (see Figure 10). Assuming the adversary makes t gates faulty in the first cycle and leaves one bit of $\widetilde{e}$ set to ‘1’ (one fault inside the circuit, $t-1$ faults on the output of the OR operations). A multivariate adversary would be able to directly set this bit to ‘0’ in the following cycle by targeting the register cell directly and, therefore, would circumvent the error detection. In order to resist against such a case, it is necessary to increase the bit size of $\widetilde{e}$ and its corresponding register until it becomes impossible to set all these bits to ‘0’ with at most t fault per clock cycle. While for this simple example it is sufficient to increase the size to $d + (t-1)$ bits (i.e., this would leave $d + (t-1) - (t-1) = d > t$ bits set to ‘1’ after the first cycle), we also have to consider other attack scenarios. Suppose that

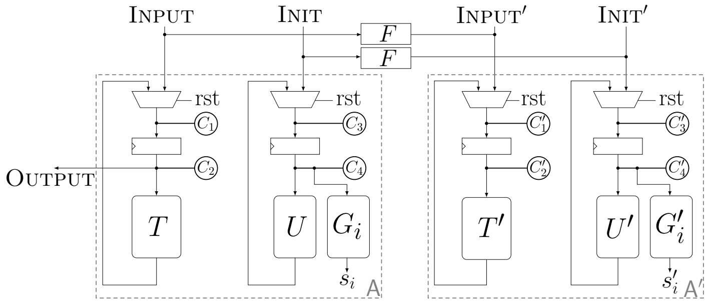


Fig. 11. Supporting the multivariate $\mathcal{M}_t^*$ model.

all bits in $\widehat{e}$ are set to '1'. Now the adversary can reduce the Hamming weight of $\widetilde{e}$ by t by targeting the output gates of the OR operation in one clock cycle. In the next clock cycle, she can again reduce it by t by targeting the register cells. Therefore, we have to increase the bit size of the error register to $t + t + 1 = d + t > 2t$ which guarantees that at least one '1' will reach the OR operation and, thus, be mapped to $d + t$ '1's before the third cycle. Hence, the OR operations (i.e., whose result are indicated by $\widetilde{e}$), the error register, and the control signal of the redundant multiplexer should change from $d - 1$ bits to $d + t = 2d - 1$ bits to support protection against the multivariate $\mathcal{M}_{t=d-1}^*$ model.

4.6 Combination with Side-Channel Countermeasures

Since our constructions make use of a binary linear code, none of the redundant functions has algebraic degree higher than their original counterpart. Therefore, application of hardware masking schemes (TI [46, 50] and DOM [30]) on our constructions would not face any trouble. However, particular attention should be paid on the consistency check module receiving the masked data to avoid side-channel leakage (see [54] for an exemplary solution). Note that it is obviously not required to mask the control logic, but all masked functions should fulfill the independence property as well. As a side note, the combination presented in [54] mixing TI and an EDC is a special case for $m = k = d = 4$. However, it does not deal with full fault coverage (i.e., the independence property has been ignored) and the control logic is excluded from the underlying fault-detection mechanism. Therefore, independent of its resistance against side-channel analysis attacks, it cannot detect all possible up to $t = d - 1$ faults, i.e., no full fault coverage against even a univariate $\mathcal{M}_t$ adversary.

5 Case Studies

To assess the overhead of our proposed methodology, we examined several case studies based on the encryption function of symmetric block ciphers with 64-bit state including PRESENT [16], LED [32], SIMON [11], GIFT [8], Midori [7], and Skinny [12]. Below we give details how different variants of Skinny are implemented. The rest of the covered algorithms are given in Appendix B. The analyses and comparisons shown here are based on hardware implementations using the IBM 130 nm ASIC standard cell library.

5.1 Skinny-64

The tweakable block cipher Skinny-64 [12] operates on a 64-bit state, and on a 64-, 128-, or 192-bit key. Depending on the key size, the number of cipher rounds is defined as 32, 36, or 40 rounds. After the state is loaded by the 64-bit plaintext, a 4-bit S-box is applied on each 4-bit chunk of the state. A part of a column of the state is XORed with a RoundConstant, and two first rows are further XORed with a 32-bit SubTweaKey. ShiftRows is the inverse of the AES ShiftRows (on 4-bit cells), and by MixColumns each column of the state is multiplied by M matrix:

$$M = \begin{pmatrix} 1\,0\,1\,1 \\ 1\,0\,0\,0 \\ 0\,1\,1\,0 \\ 1\,0\,1\,0 \end{pmatrix}.$$

The KeySchedule is formed by three variants each of which operating on a 64-bit part of the key. All variants share a nibble-wise permutation P, which is the sole operation for the first variant of the 64-bit KeySchedule. The second and the third 64-bit KeySchedule variants additionally make use of an LFSR-based operation on each nibble of the first two rows of the key after applying the permutation P.

5.2 Implementation Details

Unless otherwise stated, we focus on a round-based implementation architecture, where at every clock cycle a full encryption round is completed. In order to equip the implementation with an EDC, we first need to specify the parameters of the underlying code. The specification of an $[n, k, d]$-code is commonly defined by the largest operation of the algorithm with respect to the bit-length. Due to the 4-bit S-box of Skinny-64, the rank k is fixed to 4, and depending on the considered adversary model $\mathcal{M}_t$, the length n and distance d are defined. Below we categorize our implementations into three groups:

- $(n, d) = (8, 4)$. This implies the case with $m \geq k$ (see Figure 5(b)). The common code for this case is the extended Hamming-code $[8, 4, 4]$ (as also used in [54]).
- $(n, d) = (7, 3)/(6, 2)/(5, 2)$, i.e., with $m < k$ (see Figure 5(c)). The $[7, 4, 3]$-code is the well-known Hamming code, and $[5, 4, 2]$-code computes 1-bit parity for each 4-bit chunk. The remaining code $[6, 4, 2]$ adds one bit redundancy compared to the parity code, but its distance $d = 2$ indicates that its full fault coverage is the same as that of the parity.
- $(n, d) = (8, 2)/(12, 3)/(16, 4)$, i.e., duplication, triplication, and quadruplication respectively. We included these cases into our investigations to enable a comparison between our methodology and common and straightforward duplication schemes which provide full fault coverage considering the same adversary model.

$[8, 4, 4]$-code.
Figure 13 (in Appendix B) shows the design architecture for this case. Note that K_0, K_1, and K_2 indicate the first, second, and third 64-bit part of the key. The generator matrix G of the extended Hamming code is

$$G_{eH} = \begin{pmatrix} 1\,0\,0\,0 & 1\,1\,1\,0 \\ 0\,1\,0\,0 & 1\,1\,0\,1 \\ 0\,0\,1\,0 & 1\,0\,1\,1 \\ 0\,0\,0\,1 & 0\,1\,1\,1 \end{pmatrix} = (I_4 \mid P_{eH}), \quad F(x) = x \cdot P_{eH}.$$

As given in Section 4.4, the redundant part of the S-box operating on the redundant part of the state is derived by $F \circ S \circ F^{-1}$. Since ShiftRows is a nibble-wise permutation, it is the same as its redundant counterpart. The matrix of the MixColumns involves only 1 or 0 coefficients. With respect to the linear property of the underlying code, similar to ShiftRows the same MixColumns module is used in the redundant part of the circuit. This can be also seen in Figure 13 (in Appendix B), where the aforementioned modules are marked by SR and MC. As an important notice, the Skinny MixColumns has the special property explained in Section 4.4(§ Optimization) indicating that no extra checkpoint before the MixColumns is required.

Since the nibbles of the key are also encoded in the similar way, the KeyAddition is also done trivially on the redundancy. The same holds for the nibble-wise permutation of the key schedule (shown by P in Figure 13 in Appendix B). Similar to the S-box module, the LFSRs used in the KeySchedule of the second and third 64-bit keys K_1 and K_2 need to realize $F \circ LFSR \circ F^{-1}$. The remaining operation is the XOR with RoundConstant (shown by RC) derived from a 6-bit LFSR which is also used as the round counter defined as

$$(rc_5|rc_4|rc_3|rc_2|rc_1|rc_0) \mapsto (rc_4|rc_3|rc_2|rc_1|rc_0||rc_5 \oplus rc_4 \oplus 1). \tag{6}$$

The RoundConstant $(c_0, c_1, c_2, 0)$ is XORed to the first column of the cipher state with

$$c_0 = (rc_3|rc_2|rc_1|rc_0), \quad c_1 = (0|0|rc_5|rc_4), \quad c_2 = (0|0|1|0).$$

If the STATE of the FSM is encoded following the way it is used by c_0 and c_1, the STATE$'$ which is of $2m = 8$ bits can easily be split to make the redundant RoundConstant c'_0 and c'_1. Obviously, the last one $c'_2 = F(c_2)$.

The update function $U(.)$ of the FSM operates on 6 bits. However, the redundant counterpart $U'(.)$ operates on 8-bit STATE$'$ (see Figure 7(b)). Therefore, every component-function $U'^i(.)$ is an 8-bit to 1-bit function which makes it area-wise larger than the corresponding simple component-functions $U^i(.)$ (see Equation (6)). The FSM includes only one control signal done indicating the end of the encryption process.

Considering an $\mathcal{M}_{t=3}$ adversary model, the checkpoints are placed at the input of the S-box, at the input of the permutation modules of the KeySchedule and the single control signal done.

[7, 4, 3]-/[6, 4, 2]-/[5, 4, 2]-code.
With smaller redundancy size $m < 4$, several modules (e.g., SR, MC, P, and XORs) can solely operate on STATE$'$. The S-box and the LFSRs are excluded from this list, as it can be seen in Figure 14 (in Appendix B). The redundant counterpart of the S-box realizes $F \circ S$ and the same holds for the LFSR of the KeySchedule as $F \circ LFSR$. We used the following generator matrices in our implementations:

$$G_{[7,4,3]} = \begin{pmatrix} 1\,0\,0\,0 & 0\,1\,1 \\ 0\,1\,0\,0 & 1\,1\,1 \\ 0\,0\,1\,0 & 1\,0\,1 \\ 0\,0\,0\,1 & 1\,1\,0 \end{pmatrix}, G_{[6,4,2]} = \begin{pmatrix} 1\,0\,0\,0 & 0\,1 \\ 0\,1\,0\,0 & 1\,1 \\ 0\,0\,1\,0 & 1\,0 \\ 0\,0\,0\,1 & 1\,1 \end{pmatrix}, G_{[5,4,2]} = \begin{pmatrix} 1\,0\,0\,0 & 1 \\ 0\,1\,0\,0 & 1 \\ 0\,0\,1\,0 & 1 \\ 0\,0\,0\,1 & 1 \end{pmatrix}.$$

[2, 1, 2]-/[3, 1, 3]-/[4, 1, 4]-code.
The generator matrix of duplication is already given in Section 3.2 which is formed by $G_{dup} = (I_k \mid I_k)$. Similarly, that of triplication and quadruplication are made by three/four times repeating the identity matrix I_k. However, for the application of such schemes it is not necessary to fulfill the independence property. We instantiated every instance of the encryption function and placed the checkpoints at the CIPHERTEXT$_i$ as well as the control signals done$_i$ (see Figure 12 in Appendix B showing a general architecture of such schemes). It is noteworthy that in order to keep full fault coverage against an $\mathcal{M}_{t=d-1}$ adversary, for the given distance d we have used our-proposed consistency check module shown in Section 4.4(§ Checkpoints).

Table 1. Area (GE) and Latency (ns) comparison of our implementations considering an $\mathcal{M}_{t=d-1}$-bounded adversary with an $[n, k, d]$-code, using IBM 130 nm ASIC library.

Algorithm	Key	clock	plain		$[\mathbf{5,4,2}]$		$[\mathbf{6,4,2}]$		$[\mathbf{7,4,3}]$		$[\mathbf{8,4,4}]$		$[\mathbf{2,1,2}]$		$[\mathbf{3,1,3}]$		$[\mathbf{4,1,4}]$	
		cycles	area	lat.	area	lat.	area	lat.	area	lat.	area	lat.	area	lat.	area	lat.	area	lat.
Skinny	64	33	1243	4.11	2732	5.27	3351	5.38	4153	6.08	5041	6.43	3155	4.53	4649	5.04	6147	5.68
Skinny-serial	64	688	990	4.24	1924	5.53	2408	5.60	3383	5.75	4870	7.66	2019	4.28	3028	4.29	4042	4.78
LED-2check	64	33	1665	7.72	3667	9.17	4467	10.13	5697	11.52	7226	12.15	3968	7.69	5874	7.71	7784	7.75
LED-combine					4155	8.45	4861	8.95	6039	9.44	7716	10.40						
Skinny	128	37	1738	3.66	3640	5.16	4494	5.24	5636	6.09	6804	6.37	4128	4.34	6109	4.86	8095	5.50
Skinny-serial	128	772	1446	4.03	2778	5.42	3461	5.46	4867	5.92	6883	7.18	2906	5.55	4358	5.55	5815	5.55
LED-2check	128	49	1664	9.15	3996	9.81	4900	10.07	6359	11.66	8240	13.40	3991	9.37	5907	9.37	7822	9.36
LED-combine					4499	9.57	5264	9.17	6699	10.04	8718	12.80						
Midori	128	17	1372	7.57	3282	8.25	3942	8.16	5262	8.87	6840	10.40	3412	8.81	5029	9.51	6657	9.85
PRESENT	128	32	1767	2.93	4211	5.19	5177	5.62	6639	6.32	8219	6.71	4174	4.59	6179	5.10	8186	5.78
GIFT	128	29	1587	2.88	3824	5.11	4722	5.32	6082	6.11	7767	6.61	3847	4.33	5688	4.83	7533	5.47
SIMON	128	45	1629	2.86	3614	5.20	4487	5.27	5621	5.93	7603	6.44	3912	4.47	5785	4.97	7663	5.61
Skinny	192	41	2206	4.00	4540	5.63	5656	5.74	7119	6.34	8553	6.74	5054	4.47	7494	5.00	9940	5.62
Skinny-serial	192	856	1896	5.12	3602	5.28	4490	5.33	6321	5.76	8893	7.53	3807	4.44	5710	4.44	7618	4.76

Serial Architecture.
We additionally considered a nibble-serial architecture in our implementations of Skinny. In this fashion, which is known to provide the smallest area footprint at the cost of low throughput[10], the cipher state register is shifted one nibble at every clock cycle. Only one instance of each operational module (S-box and MixColumns) is implemented at the cost of a more complicated FSM (see Figure 15 in Appendix B). This architecture for sure reduces the area, but since at every clock cycle the state (and the key) registers are shifted, their consistency should be checked when the implementation is equipped with an EDC. Therefore, as shown in Figure 15 to Figure 18 (in Appendix B), the checkpoints are placed at all 64-bit output of the state register as well as the entire key register. This means that due to the fact that – compared to the round-based architecture – the size of the checkpoints is not reduced, the gain with respect to the area reduction is not expected to be significantly high.

5.3 Comparison

In addition to all variants of Skinny-64, we applied the aforementioned codes on round-based implementation of encryption function of LED-64, LED-128, PRESENT-128, SIMON-64, GIFT-64 and Midori-64 all with 64-bit plaintext and 128-bit key (except LED-64). The corresponding figure for each algorithm is given in Appendix B. In contrast to Skinny and Midori, the MixColumns of LED forces to place an extra checkpoint unless the S-box and the MixColumns are combined and the entire round function fulfills the independence property. Therefore, for LED we considered two variants referred as

[10] Excluding the bit-serial fashion [35].

'2check' and 'combine' respectively (for example see Figure 19 in Appendix B). In comparison, the '2check' variant leads to a smaller area overhead but with higher latency due to its extra checkpoints. With respect to other algorithms we faced several challenges when the operations do not fit into the nibble-wise fashion of the encoding, i.e., how the $[n, k, d]$-code is applied. The extreme cases include the bit-permutation of PRESENT and GIFT as well as the bit-wise shift and operations of SIMON. Considering the independence property, the redundant counterpart of such operations led to large (e.g., 12-bit to 1-bit) functions, hence high area overhead. Our implementations were synthesized using the Synopsys Design Compiler and publicly-available IBM 130 nm standard cell library. By keeping the hierarchy, we made sure that synthesizer does not corrupt the modules designed with independence property. The results are summarized in Table 1. Note that the clock cycle was not tightened allowing the synthesizer to reach the smallest area. Comparing the columns with the same distance d, it can be seen that in many cases (excluding the nibble-serial variant of Skinny[11]) our approach outperforms the duplication schemes. However, such benefits depend on the target algorithm. For instance, in almost all cases of LED128 the duplication outperforms our approach, that is the other way around in case of SIMON. The same observations can be seen in terms of latency.

As given in Section 4.5, to extend the resistance of an implementation to the corresponding multivariate adversary $\mathcal{M}_t^*$, (a) extra checkpoints should be placed at the input of every register in the design, and (b) the consistency check module should be adjusted accordingly. This is independent of the underlying functions of the cipher; the number of register bits in combination with the employed code define the additional area required for such an extension. It is noteworthy that since such extra checkpoints are placed at the registers' input, the latency of the circuit is also increased by a roughly constant value. Note that the same holds for the duplication/triplication/quadruplication techniques as well. In other words, without introducing such extra checkpoints, they are also unable to provide full fault coverage against a multivariate adversary $\mathcal{M}_t^*$. Considering such an adversary model, we summarize the area and latency figures of our implementation in Table 2 (in Appendix A). In this case, the advantage of our approach compared to duplication schemes can be seen more clearly.

6 Conclusion

Fault attacks can be easily utilized to extract sensitive information from any unprotected cryptographic implementation. Therefore, the inclusion of a dedicated countermeasure in the design process is essential and sparked numerous research contributions covering different hardening techniques. However, we have shown that the actual realization of these schemes in practice is not trivial. Many previous publications have not considered the crucial threat of fault propagation and, thus, provide only a reduced detection potential.

In this work, we have defined an adjustable adversary which takes advantages of this phenomenon and presented design strategies to cope with this new constraint. Our concepts allow the robust implementation of CED schemes in the presence of fault propagation. We defined a univariate (resp. multivariate) adversary model, in which the attacker at one (resp. every) clock cycle is able to make up to t cells faulty in the entire circuit. Accordingly, we showed how to provide security (i.e., full fault coverage) against such a powerful adversary with high precision. Furthermore, we extended our observations to the often-neglected protection of control signals and presented solutions to achieve an entirely fault-resistant architecture.

[11] This is due to the fact that the consistency check of duplication is also performed on only small 4-bit output port.

Our case studies show the efficiency of our approach for different symmetric block ciphers and highlight the effect of the chosen code on the resulting overhead. Overall, to the best of our knowledge, we presented the first secure and efficient design methodology against a realistic t-cell bounded adversary in the presence of fault propagation.

Regarding future works, a practical evaluation of the fault-resistance of our designs using actual experiments could be of great interest. This would not only increase the confidence in our methodology, but also allows to obtain a realistic estimate for the number of possible faulty cells t in practice. It is noteworthy that the fault detection ability of our constructions relies on the definition of the underlying code. Hence, the fault coverage of every module is straightforwardly obtained. However, there is an obvious lack of a simulation/verification tool to examine the fault coverage of a given design considering a certain adversary model. The available logic simulation tools have not been designed for this purpose. The scientific community would for sure benefit by having such a tool enabling verification of the claimed fault coverages.

Another important aspect which needs to be further examined is error correction. Recently, it has been demonstrated that combining a CED with state randomization (i.e., masking) does not provide sufficient protection against statistical ineffective fault attacks (SIFA) [24]. One proposed countermeasure is the inclusion of dummy rounds or other hiding techniques which can be straight-forwardly combined with our methodology. In addition, however, extending the encoded circuit with the capability to correct faulty states would raise the bar for an adversary to create the errors even further. Therefore, a combination of different techniques might provide the best results, but this requires further research especially regarding combined attacks.

Acknowledgment

The work described in this paper has been supported in part by the German Federal Ministry of Education and Research BMBF under grant number 16KIS0602 VeriSec, the European Unions Horizon 2020 program under project number 645622 PQCRYPTO, and the European Commission through the ERC project 724725 (acronym SWORD).

References

1. M. Agoyan, J. Dutertre, A. Mirbaha, D. Naccache, A. Ribotta, and A. Tria. How to flip a bit? In *IOLTS*, pages 235–239. IEEE Computer Society, 2010.
2. M. Agoyan, J. Dutertre, D. Naccache, B. Robisson, and A. Tria. When Clocks Fail: On Critical Paths and Clock Faults. In *CARDIS*, volume 6035 of *LNCS*, pages 182–193, 2010.
3. M. Agoyan, J. M. Dutertre, A. P. Mirbaha, D. Naccache, A. L. Ribotta, and A. Tria. Single-bit DFA using multiple-byte laser fault injection. In *HST*, pages 113–119, 2010.
4. K. D. Akdemir, Z. Wang, M. G. Karpovsky, and B. Sunar. Design of Cryptographic Devices Resilient to Fault Injection Attacks Using Nonlinear Robust Codes. In *Fault Analysis in Cryptography*, pages 171–199. Springer, 2012.
5. C. Ananiadis, A. Papadimitriou, D. Hély, V. Beroulle, P. Maistri, and R. Leveugle. On the development of a new countermeasure based on a laser attack RTL fault model. In *DATE*, pages 445–450. IEEE, 2016.
6. S. Azzi, B. Barras, M. Christofi, and D. Vigilant. Using linear codes as a fault countermeasure for nonlinear operations: application to AES and formal verification. *J. Cryptographic Engineering*, 7(1):75–85, 2017.
7. S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, and F. Regazzoni. Midori: A Block Cipher for Low Energy. In *ASIACRYPT*, volume 9453 of *LNCS*, pages 411–436. Springer, 2015.
8. S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo. GIFT: A Small Present - Towards Reaching the Limit of Lightweight Encryption. In *CHES*, volume 10529 of *LNCS*, pages 321–345. Springer, 2017.

9. H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The Sorcerer's Apprentice Guide to Fault Attacks. *IACR Cryptology ePrint Archive*, 2004:100, 2004.
10. A. Battistello and C. Giraud. Fault Cryptanalysis of CHES 2014 Symmetric Infective Countermeasure. *IACR Cryptology ePrint Archive*, 2015:500, 2015.
11. R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. The SIMON and SPECK lightweight block ciphers. In *DAC*, pages 175:1–175:6. ACM, 2015.
12. C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In *CRYPTO*, volume 9815 of *LNCS*, pages 123–153. Springer, 2016.
13. G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri. Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard. *IEEE Trans. Computers*, 52(4):492–505, 2003.
14. E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *CRYPTO*, volume 1294 of *LNCS*, pages 513–525. Springer, 1997.
15. R. E. Blahut. *Algebraic codes for data transmission.* Cambridge Univ. Press, 2003.
16. 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 *CHES*, volume 4727 of *LNCS*, pages 450–466. Springer, 2007.
17. D. Boneh, R. A. DeMillo, and R. J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract). In *EUROCRYPT*, volume 1233 of *LNCS*, pages 37–51. Springer, 1997.
18. J. Bringer, C. Carlet, H. Chabanne, S. Guilley, and H. Maghrebi. Orthogonal Direct Sum Masking - A Smartcard Friendly Computation Paradigm in a Code, with Builtin Protection against SCA and Fault Attacks. In *WISTP*, volume 8501 of *LNCS*, pages 40–56, 2014.
19. C. Carlet and S. Guilley. Complementary dual codes for counter-measures to side-channel attacks. *Adv. in Math. of Comm.*, 10(1):131–150, 2016.
20. R. B. Carpi, S. Picek, L. Batina, F. Menarini, D. Jakobovic, and M. Golub. Glitch It If You Can: Parameter Search Strategies for Successful Fault Injection. In *CARDIS*, volume 8419 of *LNCS*, pages 236–252. Springer, 2013.
21. C. Clavier and A. Wurcker. Reverse Engineering of a Secret AES-like Cipher by Ineffective Fault Analysis. In *FDTC*, pages 119–128. IEEE Computer Society, 2013.
22. T. D. Cnudde and S. Nikova. More Efficient Private Circuits II through Threshold Implementations. In *FDTC*, pages 114–124. IEEE Computer Society, 2016.
23. F. Courbon, P. Loubet-Moundi, J. J. A. Fournier, and A. Tria. Adjusting Laser Injections for Fully Controlled Faults. In *COSADE*, volume 8622 of *LNCS*, pages 229–242, 2014.
24. C. Dobraunig, M. Eichlseder, H. Gross, S. Mangard, F. Mendel, and R. Primas. Statistical ineffective fault attacks on masked aes with fault countermeasures. Cryptology ePrint Archive, Report 2018/357, 2018. `https://eprint.iacr.org/2018/357`.
25. C. Dobraunig, M. Eichlseder, T. Korak, S. Mangard, F. Mendel, and R. Primas. Exploiting ineffective fault inductions on symmetric cryptography. Cryptology ePrint Archive, Report 2018/071, 2018. `https://eprint.iacr.org/2018/071`.
26. T. Fuhr, É. Jaulmes, V. Lomné, and A. Thillard. Fault Attacks on AES with Faulty Ciphertexts Only. In *FDTC*, pages 108–118. IEEE Computer Society, 2013.
27. G. Gaubatz, B. Sunar, and M. G. Karpovsky. Non-linear Residue Codes for Robust Public-Key Arithmetic. In *FDTC*, volume 4236 of *LNCS*, pages 173–184. Springer, 2006.
28. N. F. Ghalaty, B. Yuce, M. M. I. Taha, and P. Schaumont. Differential Fault Intensity Analysis. In *FDTC*, pages 49–58. IEEE Computer Society, 2014.
29. D. Giot, P. Roche, G. Gasiot, J. L. Autran, and R. Harboe-Sorensen. Heavy ion testing and 3D simulations of Multiple Cell Upset in 65nm standard SRAMs. In *European Conference on Radiation and Its Effects on Components and Systems*, pages 1–6, 2007.
30. H. Groß, S. Mangard, and T. Korak. An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order. In *CT-RSA 2017*, volume 10159 of *LNCS*, pages 95–112. Springer, 2017.
31. S. Guilley, L. Sauvage, J. Danger, N. Selmane, and R. Pacalet. Silicon-level Solutions to Counteract Passive and Active Attacks. In *FDTC*, pages 3–17. IEEE, 2008.
32. J. Guo, T. Peyrin, A. Poschmann, and M. J. B. Robshaw. The LED Block Cipher. In *CHES*, volume 6917 of *LNCS*, pages 326–341. Springer, 2011.

33. X. Guo, D. Mukhopadhyay, C. Jin, and R. Karri. Security analysis of concurrent error detection against differential fault analysis. *J. Cryptographic Eng.*, 5(3):153–169, 2015.
34. Y. Ishai, M. Prabhakaran, A. Sahai, and D. Wagner. Private Circuits II: Keeping Secrets in Tamperable Circuits. In *EUROCRYPT*, volume 4004 of *LNCS*, pages 308–327, 2006.
35. J. Jean, A. Moradi, T. Peyrin, and P. Sasdrich. Bit-Sliding: A Generic Technique for Bit-Serial Implementations of SPN-based Primitives - Applications to AES, PRESENT and SKINNY. In *CHES*, volume 10529 of *LNCS*, pages 687–707. Springer, 2017.
36. M. G. Karpovsky, K. J. Kulikowski, and A. Taubin. Robust Protection against Fault-Injection Attacks on Smart Cards Implementing the Advanced Encryption Standard. In *DSN*, pages 93–101. IEEE Computer Society, 2004.
37. R. Karri, K. Wu, P. Mishra, and Y. Kim. Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 21(12):1509–1517, 2002.
38. K. J. Kulikowski, M. G. Karpovsky, and A. Taubin. Robust codes and robust, fault-tolerant architectures of the Advanced Encryption Standard. *Journal of Systems Architecture*, 53(2-3):139–149, 2007.
39. K. J. Kulikowski, Z. Wang, and M. G. Karpovsky. Comparative Analysis of Robust Fault Attack Resistant Architectures for Public and Private Cryptosystems. In *FDTC*, pages 41–50. IEEE Computer Society, 2008.
40. P. Loubet-Moundi, D. Vigilant, and F. Olivier. Static Fault Attacks on Hardware DES Registers. *IACR Cryptology ePrint Archive*, 2011:531, 2011.
41. F. J. MacWilliams and N. J. A. Sloane. *The theory of error correcting codes*. North-Holland mathematical library. North-Holland Pub. Co. New York, 1977.
42. T. Malkin, F. Standaert, and M. Yung. A Comparative Cost/Security Analysis of Fault Attack Countermeasures. In *FDTC*, volume 4236 of *LNCS*, pages 159–172. Springer, 2006.
43. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks - Revealing the Secrets of Smart Cards*. Springer, 2007.
44. A. Moradi, O. Mischke, C. Paar, Y. Li, K. Ohta, and K. Sakiyama. On the Power of Fault Sensitivity Analysis and Collision Side-Channel Attacks in a Combined Setting. In *CHES*, volume 6917 of *LNCS*, pages 292–311. Springer, 2011.
45. G. D. Natale, M. Flottes, and B. Rouzeyre. An On-Line Fault Detection Scheme for SBoxes in Secure Circuits. In *IOLTS*, pages 57–62, 2007.
46. S. Nikova, V. Rijmen, and M. Schläffer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. *J. Cryptology*, 24(2):292–321, 2011.
47. V. Ocheretnij, G. Kouznetsov, R. Karri, and M. Gössel. On-Line Error Detection and BIST for the AES Encryption Algorithm with Different S-Box Implementations. In *IOLTS*, pages 141–146, 2005.
48. S. Patranabis, A. Chakraborty, P. H. Nguyen, and D. Mukhopadhyay. A Biased Fault Attack on the Time Redundancy Countermeasure for AES. In *COSADE*, volume 9064 of *LNCS*, pages 189–203. Springer, 2015.
49. J.-J. Quisquater and D. Samyde. Eddy current for magnetic analysis with active sensor. In *Proceedings of Esmart*, 2002.
50. O. Reparaz, B. Bilgin, S. Nikova, B. Gierlichs, and I. Verbauwhede. Consolidating Masking Schemes. In *CRYPTO 2015*, volume 9215 of *LNCS*, pages 764–783. Springer, 2015.
51. O. Reparaz, L. D. Meyer, B. Bilgin, V. Arribas, S. Nikova, V. Nikov, and N. Smart. CAPA: The Spirit of Beaver against Physical Attacks. Cryptology ePrint Archive, Report 2017/1195, 2017.
52. C. Roscian, A. Sarafianos, J. Dutertre, and A. Tria. Fault Model Analysis of Laser-Induced Faults in SRAM Memory Cells. In *FDTC*, pages 89–98. IEEE Computer Society, 2013.
53. F. Schellenberg, M. Finkeldey, B. Richter, M. Schapers, N. Gerhardt, M. Hofmann, and C. Paar. On the Complexity Reduction of Laser Fault Injection Campaigns Using OBIC Measurements. In *FDTC*, pages 14–27. IEEE Computer Society, 2015.
54. T. Schneider, A. Moradi, and T. Güneysu. ParTI - Towards Combined Hardware Countermeasures Against Side-Channel and Fault-Injection Attacks. In *CRYPTO*, volume 9815 of *LNCS*, pages 302–332. Springer, 2016.
55. O. Seker, T. Eisenbarth, and R. Steinwandt. Extending Glitch-Free Multiparty Protocols to Resist Fault Injection Attacks. Cryptology ePrint Archive, Report 2017/269, 2017.

56. N. Selmane, S. Guilley, and J. Danger. Practical Setup Time Violation Attacks on AES. In *EDCC-7*, pages 91–96, 2008.
57. B. Selmke, J. Heyszl, and G. Sigl. Attack on a DFA Protected AES by Simultaneous Laser Fault Injections. In *FDTC*, pages 36–46. IEEE Computer Society, 2016.
58. S. P. Skorobogatov and R. J. Anderson. Optical Fault Induction Attacks. In *CHES*, volume 2523 of *LNCS*, pages 2–12. Springer, 2002.
59. C. Yen and B. Wu. Simple Error Detection Methods for Hardware Implementation of Advanced Encryption Standard. *IEEE Trans. Computers*, 55(6):720–731, 2006.
60. B. Yuce, N. F. Ghalaty, H. Santapuri, C. Deshpande, C. Patrick, and P. Schaumont. Software Fault Resistance is Futile: Effective Single-Glitch Attacks. In *FDTC*, pages 47–58, 2016.
61. B. Yuce, N. F. Ghalaty, and P. Schaumont. TVVF: Estimating the vulnerability of hardware cryptosystems against timing violation attacks. In *HOST*, pages 72–77. IEEE, 2015.

Appendix

A Performance Figures against $\mathcal{M}_t^*$ Adversary

Table 2. Area (GE) and Latency (ns) comparison of our implementations considering an $\mathcal{M}_{t=d-1}^*$-bounded adversary with an $[n, k, d]$-code, using IBM 130 nm ASIC library.

Algorithm	**Key**	**clock**	**plain**		**[5, 4, 2]**		**[6, 4, 2]**		**[7, 4, 3]**		**[8, 4, 4]**		**[2, 1, 2]**		**[3, 1, 3]**		**[4, 1, 4]**	
		cycles	area	lat.	area	lat.	area	lat.	area	lat.	area	lat.	area	lat.	area	lat.	area	lat.
Skinny	64	33	1243	4.11	3192	7.91	3968	7.97	5085	8.25	6294	8.94	3666	7.59	5592	8.56	7525	8.87
Skinny-serial	64	688	990	4.24	2250	8.71	2900	9.03	4115	9.06	5855	11.55	2389	8.81	3765	9.92	5149	10.00
LED-2check	64	33	1665	7.72	3986	12.97	4904	12.31	6316	13.41	8086	14.52	4290	11.28	6448	11.81	8613	12.28
LED-combine					4508	12.57	5284	12.52	6720	13.47	8569	13.47						
Skinny	128	37	1738	3.66	4236	7.84	5320	8.73	6879	8.85	8477	9.60	4808	8.02	7385	9.13	9970	9.20
Skinny-serial	128	772	1446	4.03	3228	9.26	4140	9.57	5895	9.94	8257	11.56	3448	9.08	5434	10.03	7427	10.29
LED-2check	128	49	1664	9.15	4320	13.51	5315	13.35	6972	13.71	9099	16.18	4365	12.21	6557	13.35	8758	13.61
LED-combine					4813	13.11	5729	12.46	7359	12.96	9637	15.89						
Midori	128	17	1372	7.57	3615	11.18	4358	11.42	5891	11.52	7693	14.30	3761	12.62	5656	13.46	7558	13.75
PRESENT	128	32	1767	2.93	4792	7.83	6015	8.53	7899	8.87	9896	9.37	4856	7.13	7457	8.26	10067	8.59
GIFT	128	29	1587	2.88	4420	7.49	5548	8.43	7325	8.56	9432	9.01	4517	6.98	6948	7.91	9386	8.28
SIMON	128	45	1629	2.86	4211	7.28	5311	7.86	6866	8.03	9277	9.97	4581	6.77	7044	7.73	9515	8.03
Skinny	192	41	2206	4.00	5272	7.69	6690	8.26	8676	8.79	10640	9.34	5890	7.92	9092	8.87	12300	9.17
Skinny-serial	192	856	1896	5.12	4219	8.55	5400	9.03	7698	9.21	10713	11.42	4495	8.42	7084	9.67	9679	9.93

B Implementation Figures

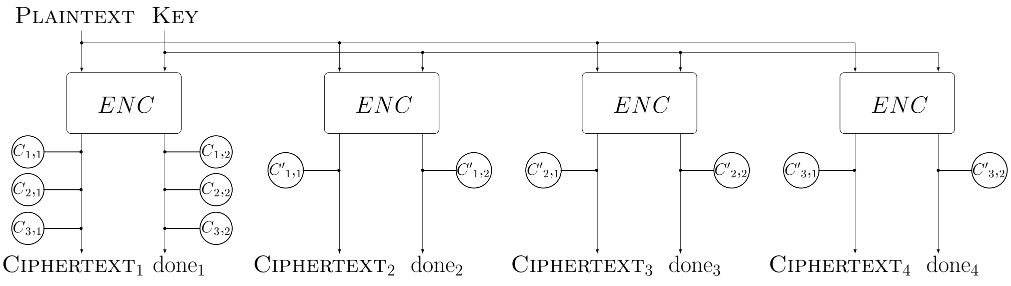


Fig. 12. Duplication, triplication, and quadruplication concept.

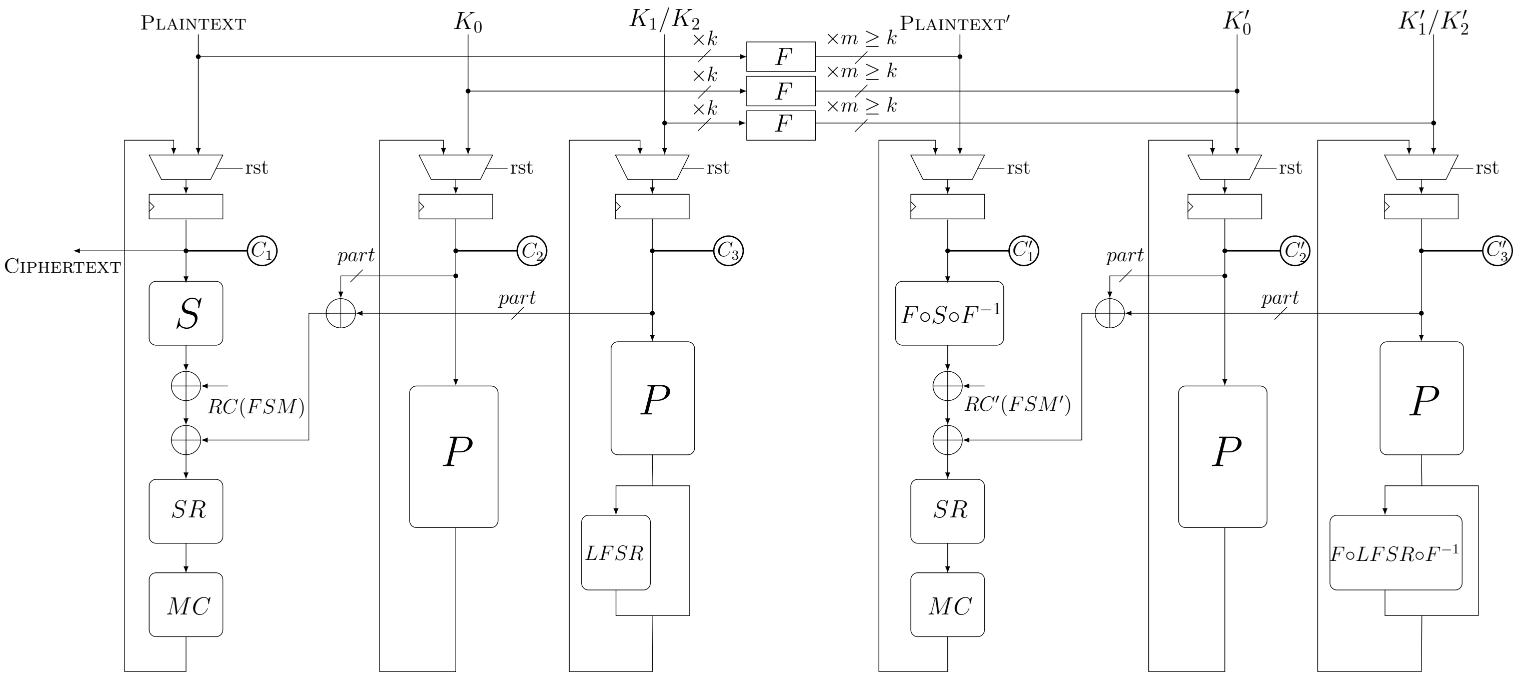


Fig. 13. Skinny, round-based, $m \geq k$.

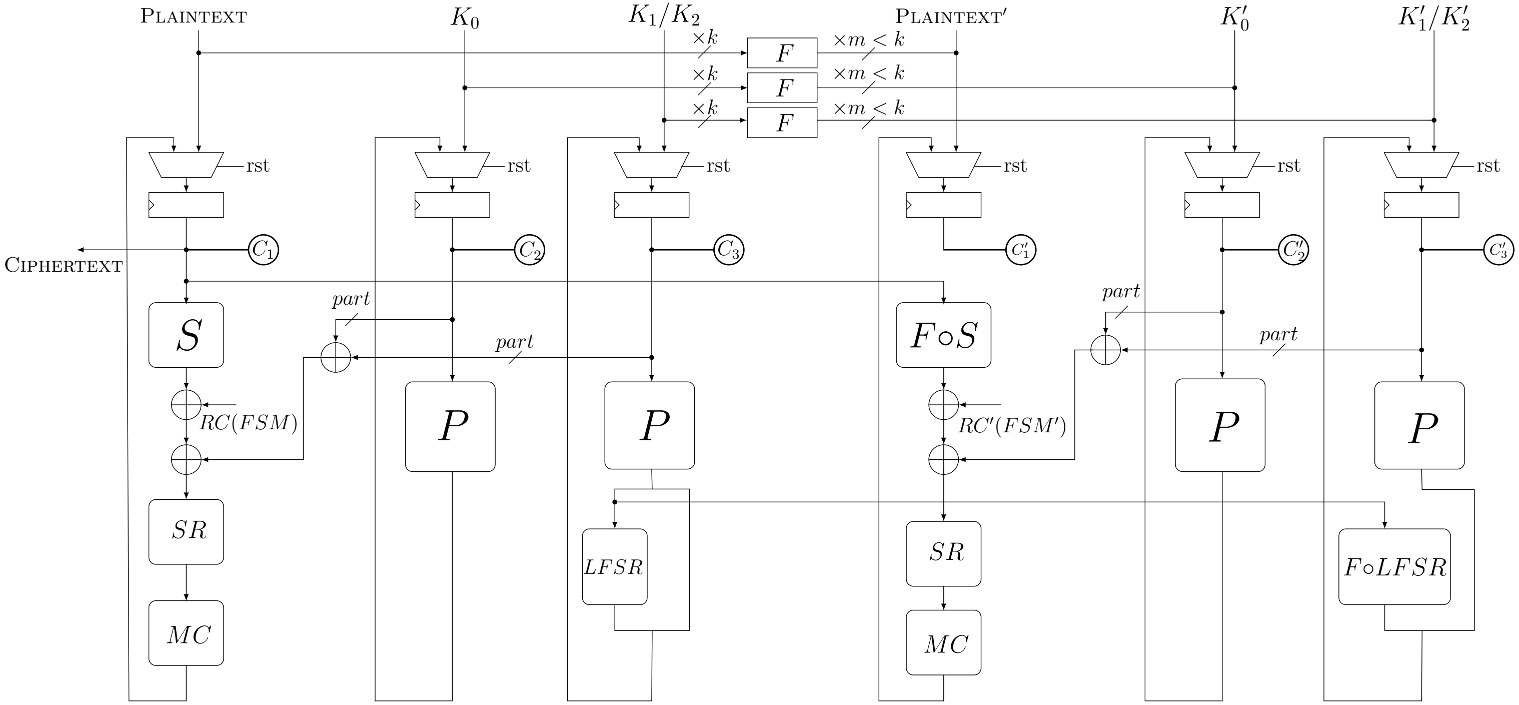


Fig. 14. Skinny, round-based, $m < k$.

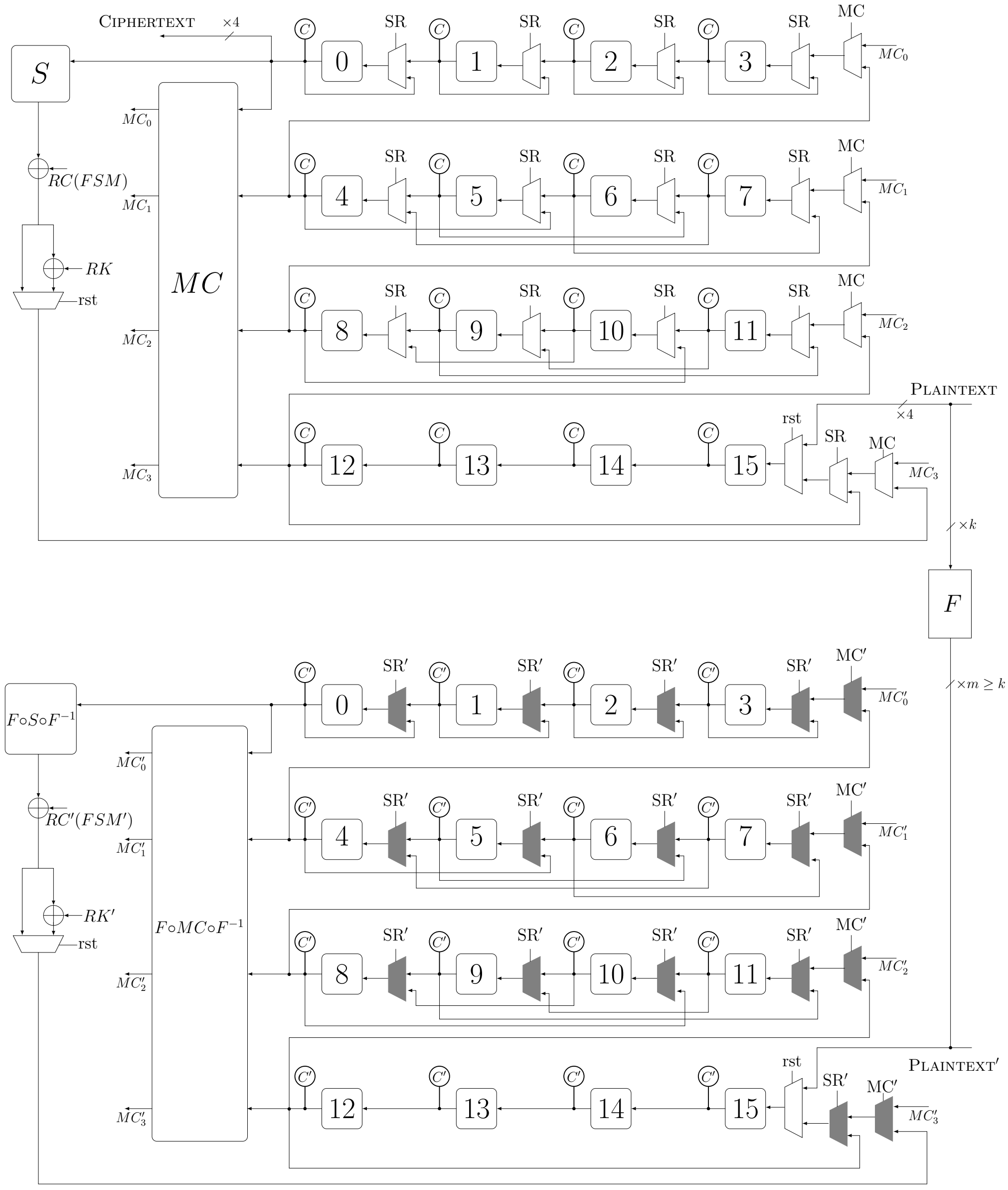


Fig. 15. Skinny, nibble-serial, data path, $m \geq k$.

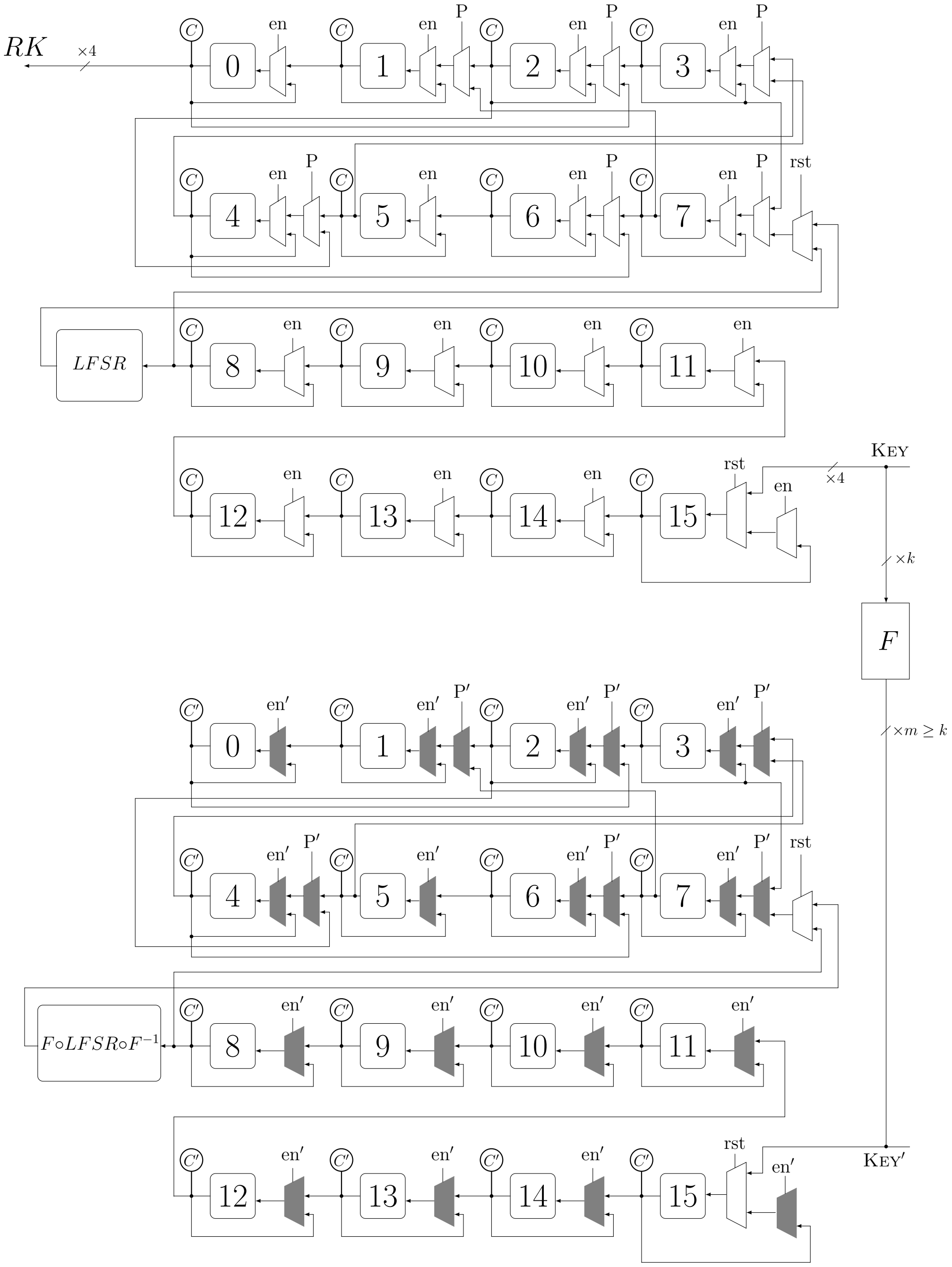


Fig. 16. Skinny, nibble-serial, key path, $m \geq k$.

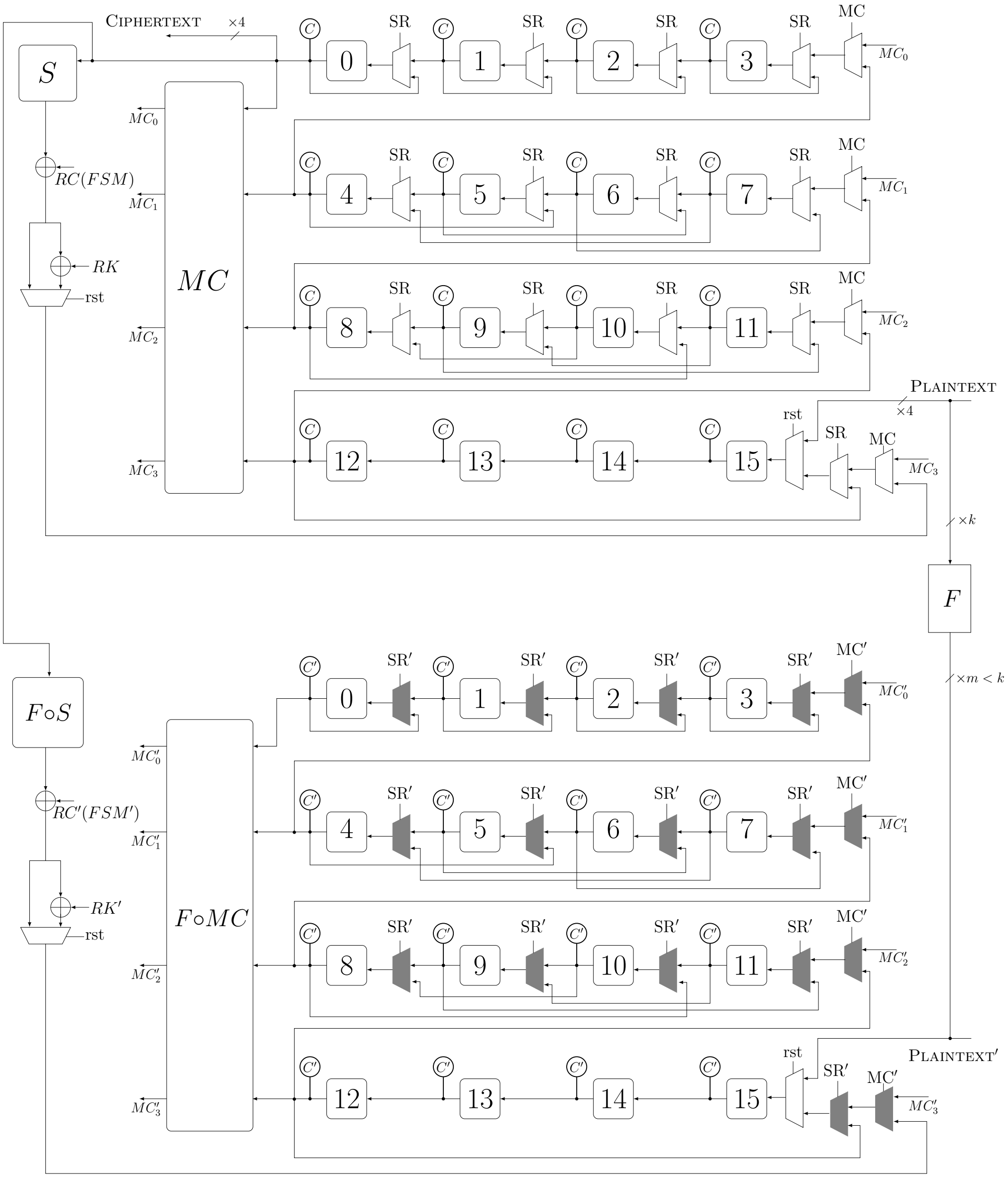


Fig. 17. Skinny, nibble-serial, data path, $m < k$.

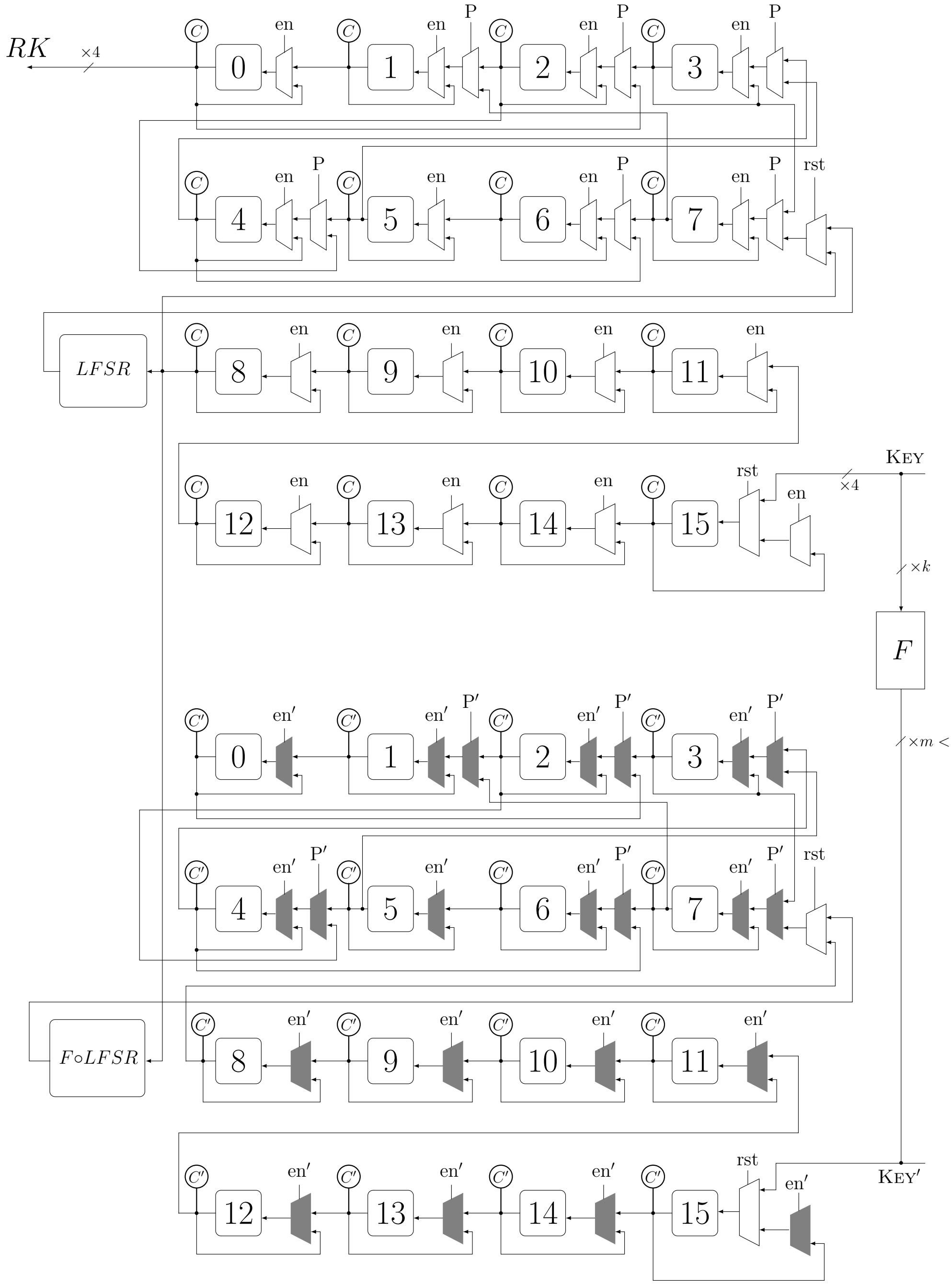
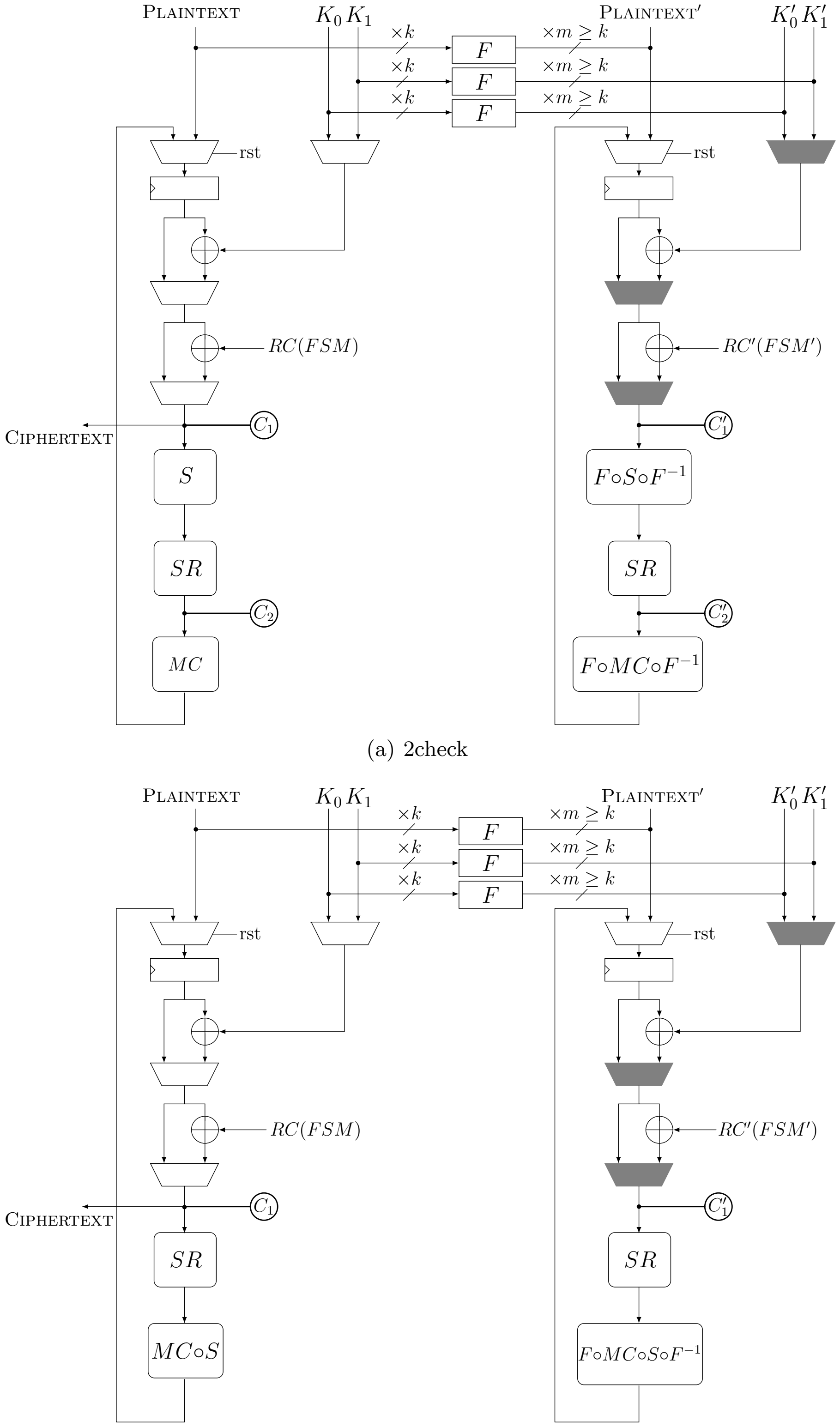


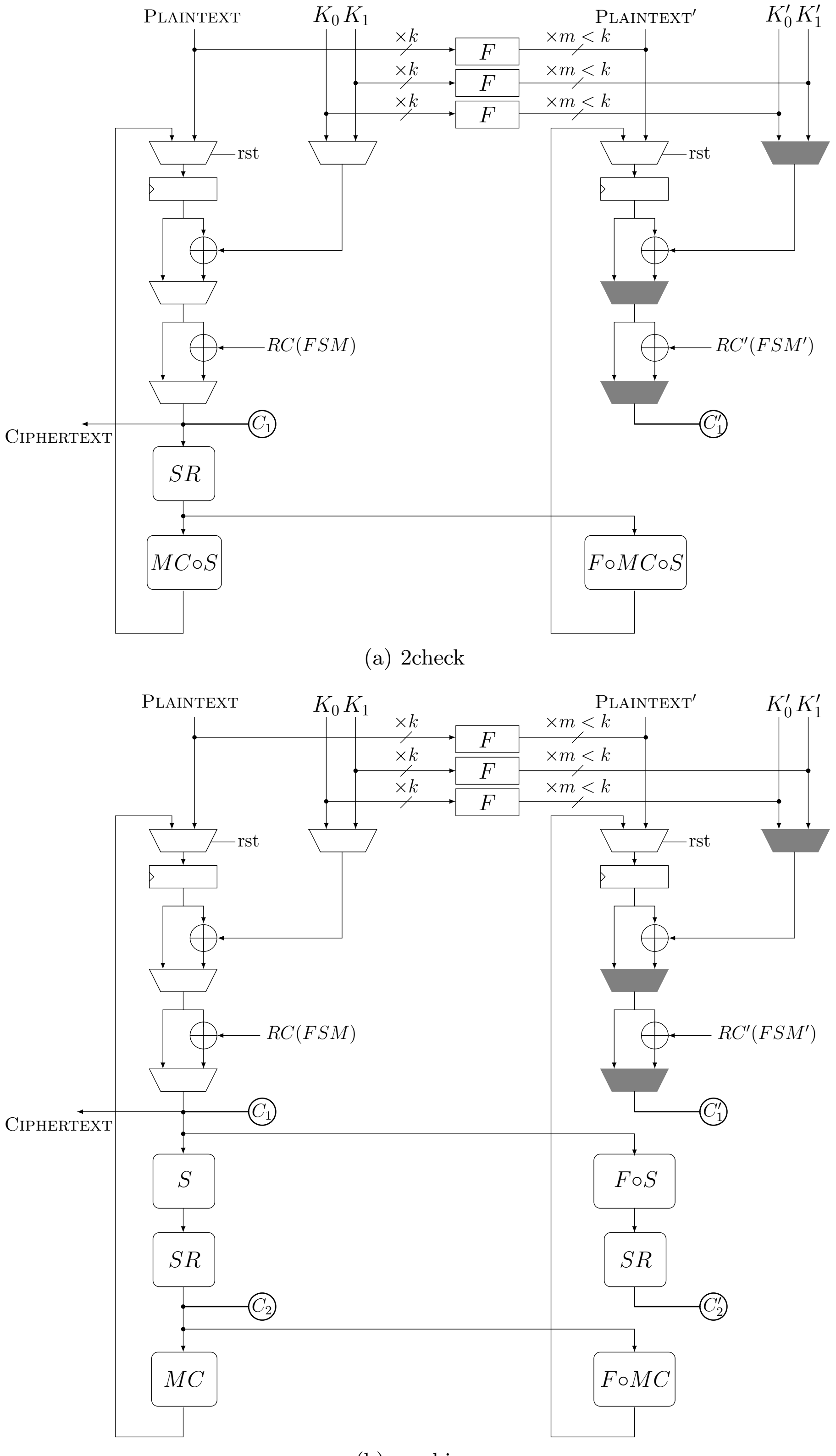
Fig. 18. Skinny, nibble-serial, key path, $m < k$.



(a) 2check

(b) combine

Fig. 19. LED, round-based, $m \geq k$.



(a) 2check

(b) combine

Fig. 20. LED, round-based, $m < k$.

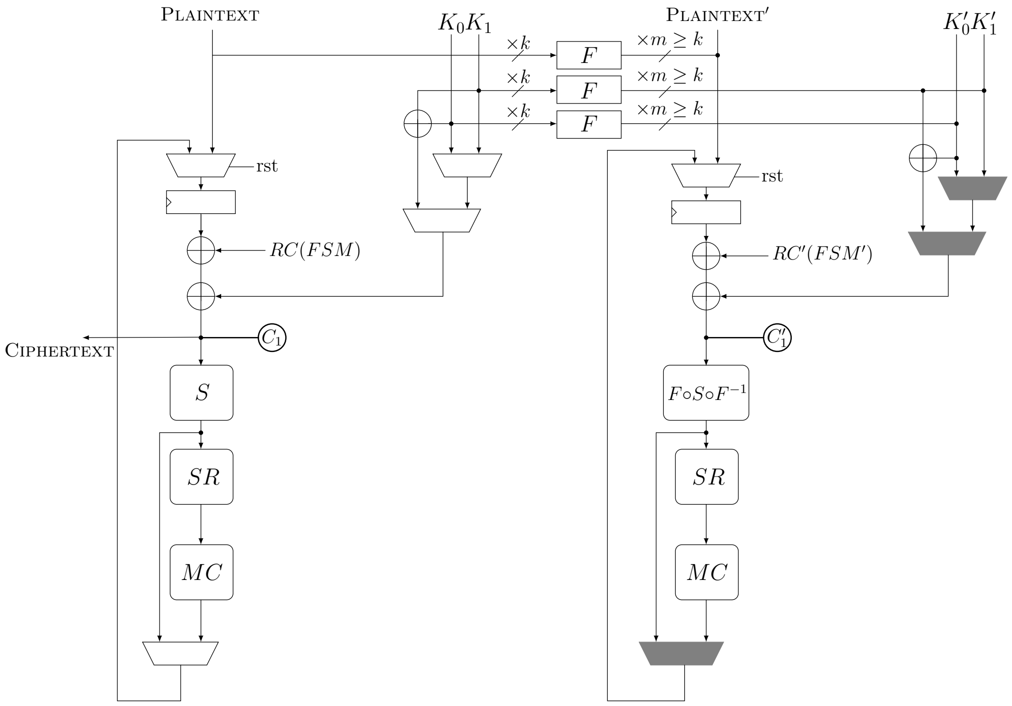


Fig. 21. Midori, round-based, $m \geq k$.

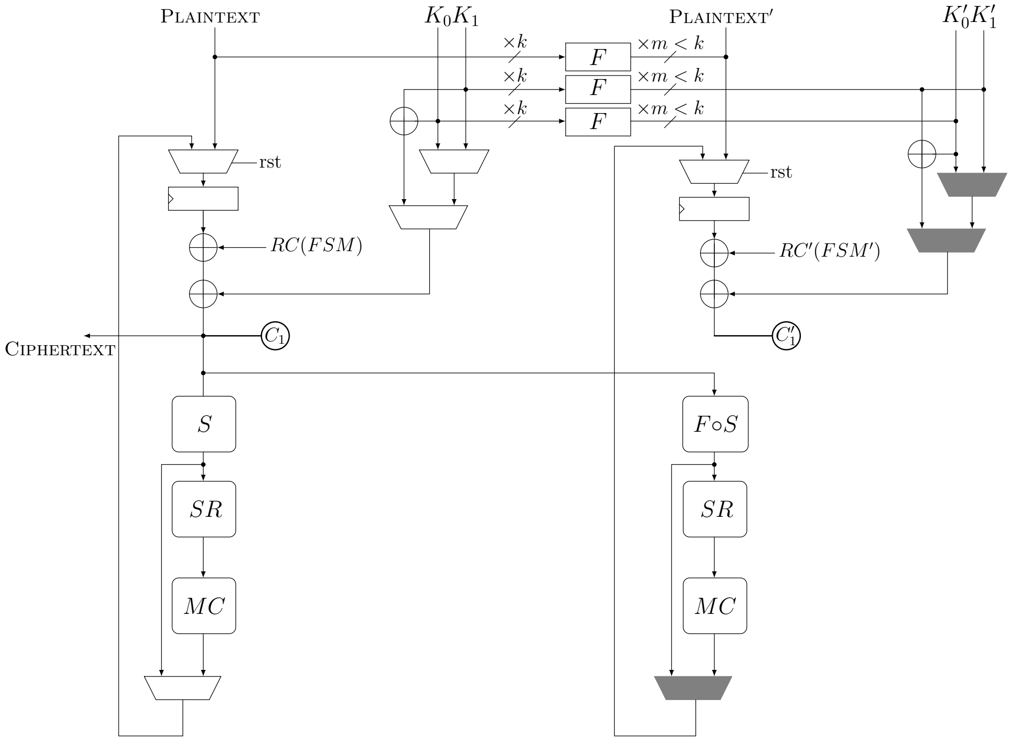


Fig. 22. Midori, round-based, $m < k$.

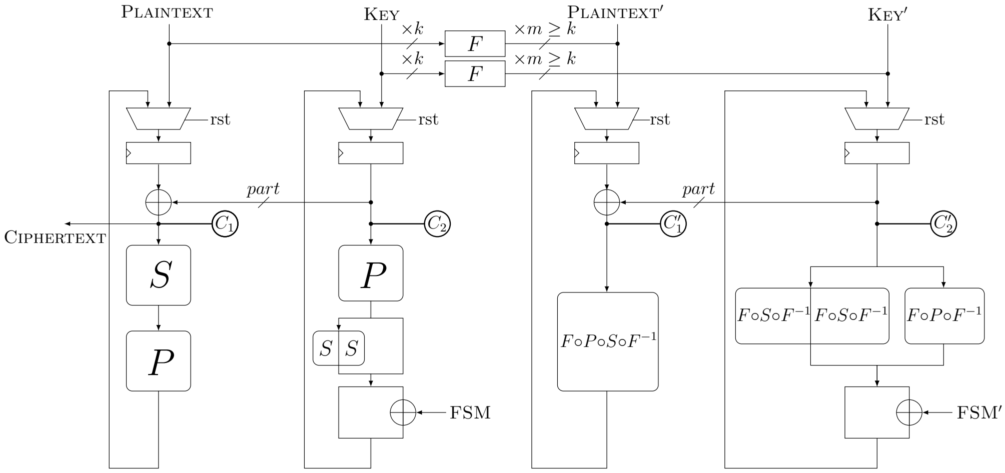


Fig. 23. PRESENT, round-based, $m \geq k$.

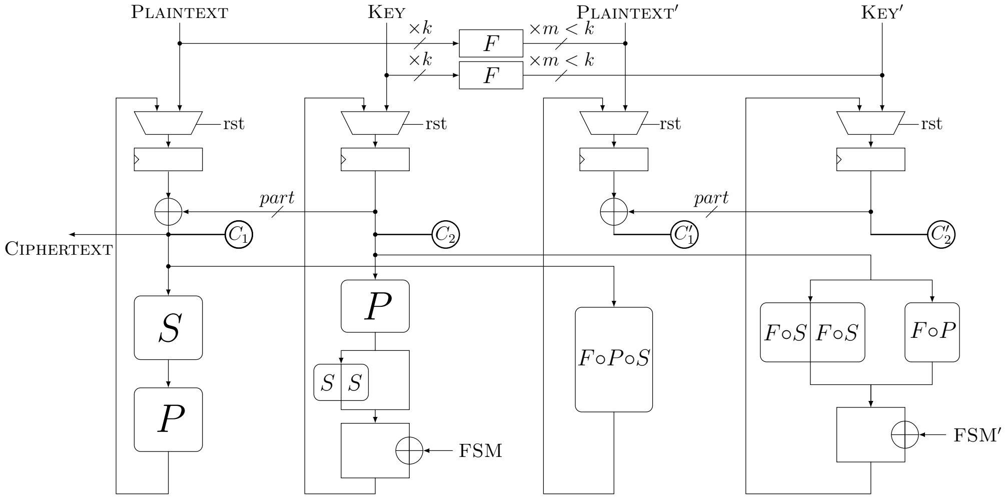


Fig. 24. PRESENT, round-based, $m < k$.

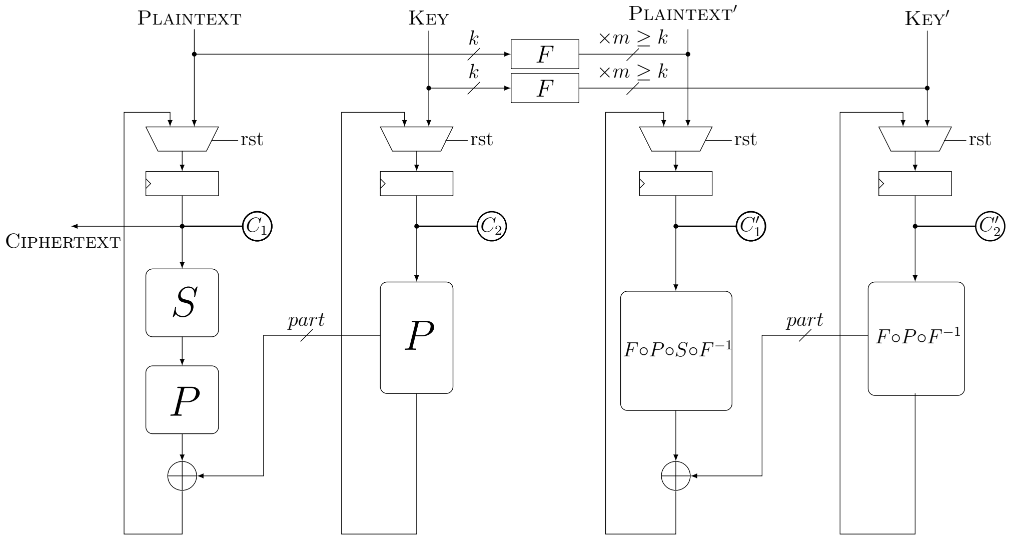


Fig. 25. GIFT, round-based, $m \geq k$.

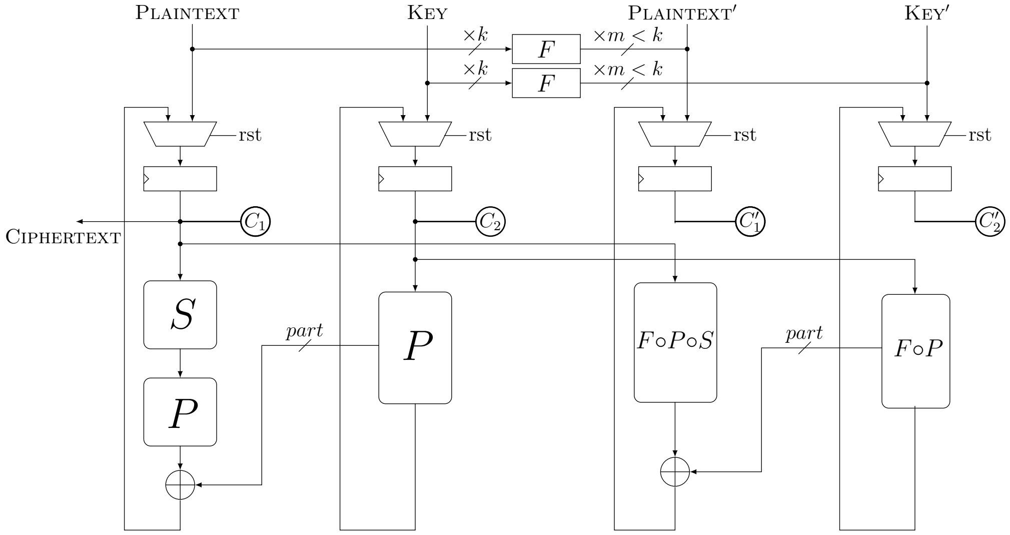


Fig. 26. GIFT, round-based, $m < k$.

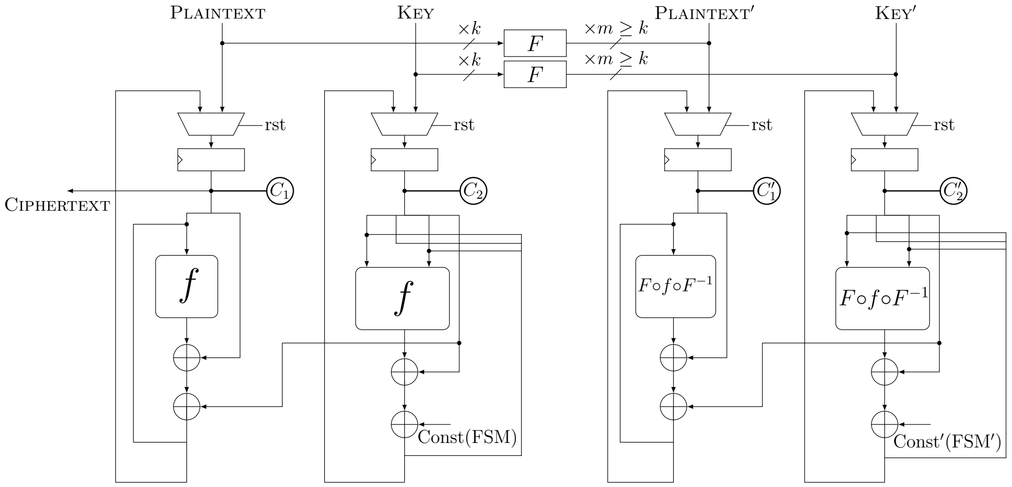


Fig. 27. SIMON, round-based, $m \geq k$.

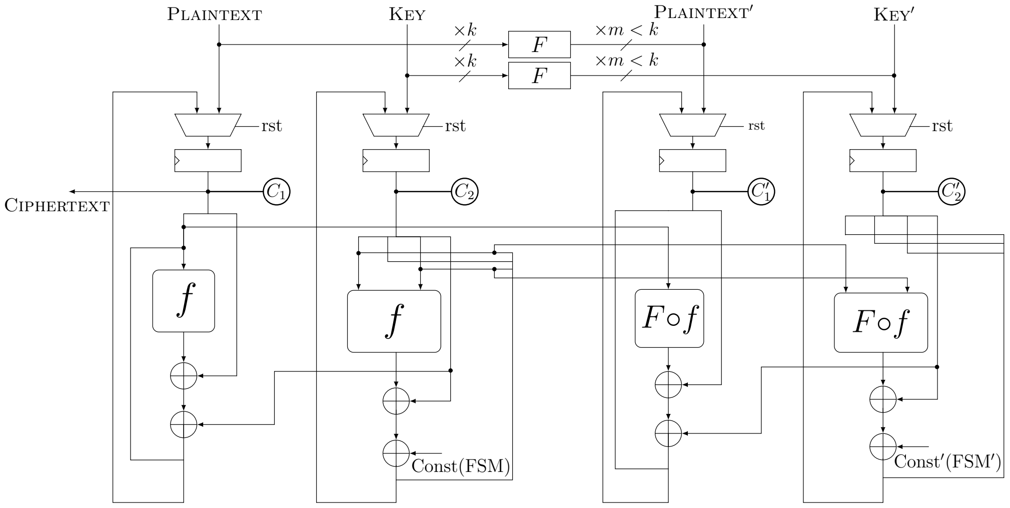


Fig. 28. SIMON, round-based, $m < k$.