Impeccable Circuits

Anita Aghaie, Amir Moradi, Shahram Rasoolzadeh,
Aein Rezaei Shahmirzadi, Falk Schellenberg, Tobias Schneider

Abstract—By injecting faults, active physical attacks pose serious threats to cryptographic hardware where Concurrent Error Detection (CED) schemes are promising countermeasures. They are usually based on an Error-Detecting Code (EDC) which enables detecting certain injected faults depending on the specification of the underlying code. Here, we propose a methodology to enable correct, practical, and robust implementation of code-based CEDs. We show that straightforward hardware implementations of given code-based CEDs can suffer from severe vulnerabilities, not providing the desired protection level. In particular, propagation of faults into combinatorial logic is often ignored in security evaluation of these schemes. First, we formally define this detrimental effect and demonstrate its destructive impact. Second, we introduce an implementation strategy to limit the fault propagation effect. Third, in contrast to many other works where the fault coverage is the main focus, we present a detailed implementation strategy which can guarantee the detection of any fault covered by the underlying EDC. This holds for any time of the computation and any location in the circuit, both in data processing and control unit. In short, we provide practical guidelines how to construct efficient CED schemes with arbitrary EDCs to achieve the desired protection level. We practically evaluate the efficiency of our methodology by case studies covering different symmetric block ciphers and various linear EDCs.

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 an 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-injection attacks, first introduced by Boneh et al. [1]. 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.

An intuitive countermeasure to such attacks is to introduce redundancy by calculating twice, either in parallel (area redundancy or *duplication*) or consecutively (time redundancy) [2], [3]. When a fault is detected, the output is omitted or sensitive data is destroyed. Since the consistency of information is checked simultaneously with the computation, such schemes are usually denoted as Concurrent Error Detection (CED).

Error Detecting Codes (EDCs) seem to be a promising approach to counter strong adversaries as they can be easily adjusted by increasing the minimum 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 as 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.

Contributions. In this paper, we present a methodology which enables a secure and practical implementation of code-based CED schemes in the presence of *fault propagation*, which we first formally define 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 hardware circuits in such a way that the detection of faults fitting into the considered model is guaranteed. We further cover every signal and component in

• *Anita Aghaie, Amir Moradi, Shahram Rasoolzadeh, Aein Rezaei Shahmirzadi, and Falk Schellenberg are with the Ruhr University Bochum, Horst Görtz Institute for IT-Security, Germany.*
E-mail: firstname.lastname@rub.de
Tobias Schneider is with NXP Semiconductors Austria GmbH. The majority of his contribution was performed while he was with Ruhr University Bochum and UC Louvain. E-mail: tobias.schneider-a7a@rub.de

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 Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972 and the project 406956718 SuCCESS, the European Unions Horizon 2020 program under project number 645622 PQCRYPTO, and the European Commission through the ERC project 724725 (acronym SWORD).

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. We in fact 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 [4], Skinny [5], Midori [6], GIFT [7], LED [8], SIMON [9] and AES [10] and 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 Works. There is an extensive body of work related to the design and implementation of CEDs. In some, the problem of fault propagation was already identified and some basic countermeasures were discussed. Below, we briefly recall related works and indicate their limitations.

Parity is often used as an EDC for CED schemes. The authors of [11] identified the fault propagation issue and as a solution suggested to divide the circuit, yet only within the context of parity-based schemes. In further previously-published articles [12], 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 [13], the authors explored how EDCs can be combined with Threshold Implementation (TI) [14] to construct an efficient hardware design resistant against both fault and side-channel attacks. Private Circuits II [15] 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 [16]. Recently, two new combined countermeasures based on multiparty computation have been proposed in [17], [18]. Practical investigation of [17] which is based on a software implementation shows a high overhead. For CAPA [18], the authors introduce a new formal adversary model including both active and passive attacks. Their approach is not affected by fault propagation, as it relies on the hardness of forging a valid MAC tag for fault resistance. The problem of fault propagation was also discussed in [19], where Timing Violation Vulnerability Factor (TVVF) as a metric to evaluate the security of a given circuit was defined. 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

For fault 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 [1]. Physical means to inject a fault include tampering with the supply voltage [20] or the clock signal [21]. Both relate to timing violations while strong electromagnetic pulses [22] can affect the target's execution as well. In contrast, optical fault injection [23] using laser beams can scale down the focus to a single transistor. Advanced optical setup can even target multiple transistors independently [24].

In practice, all physical fault injection techniques incorporate many parameters, e.g., the timing (clock cycle), the physical intensity, the duration of the effect, the location (x/y) on the device and even the distance (EM) or focal plan (laser). This already lead to various approaches trying to limit the parameters [25], [26].

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'. Faults sometimes are modeled as bit-flip, which 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 [27], [28]. 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.

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) [29]. However, there are multiple more "exotic" approaches that differ in certain aspects or requirements: Fault Sensitivity Analysis [30], Differential Fault Intensity Analysis [27], Statistical Fault Attacks [28], [31], etc. Using one or another of such attacks, the implementations of different ciphers were found to be vulnerable. In fact, DFA can be seen as a form of differential cryptanalysis on some last rounds of the cipher defined by a particular fault model.

An obvious generic countermeasure is to introduce some form of redundancy. This translates to repeating e.g., the encryption for time redundancy, or multiple encryptions in parallel for area redundancy. More sophisticated approaches employ coding schemes instead of plain redundancy [12], [32], [33]. All CED schemes commonly check whether indeed no fault occurred to enable the output.

2.2 Concurrent Error Detection Schemes

As depicted in Figure 1, a CED scheme usually includes 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. The result of A and A' are checked in module 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.

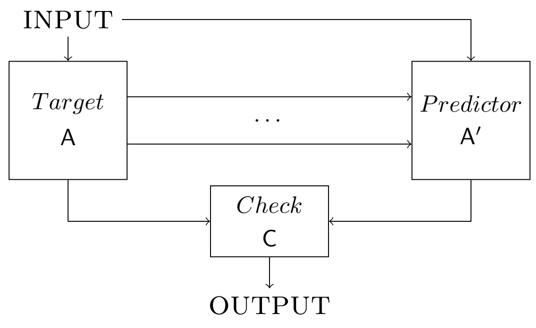


Fig. 1. Basic Structure of concurrent error detection schemes.

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., [34]. 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.

2.3 Error Detecting Codes

EDCs are an essential aspect of information theory and are often used in CED schemes. In the following, we introduce some notions [35] related to linear codes which are relevant to our work.

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

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*

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

where $wt : \mathbb{F}_2^n \mapsto \mathbb{N}$ *denotes the 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*.*

Due to the structure of the generator matrix of systematic codes, each codeword c contains the message x padded by **check bits** x', i.e., $c : \langle x, x' \rangle$. The check bits can be easily generated using the matrix P as $x' = x \cdot P$. This 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.

Example 1 (Parity)**.** *As a common approach CED scheme [32], [33], the check bits* x' *consist of only one additional bit, and the required extra logic is rather small*[1]*. This leads to a* $[k + 1, k, 2]$*-code with an error-detecting capability of* $u = 1$ *bit.*

Example 2 (Multiple Executions)**.** *Another common CED schemes is to simply run the target algorithm multiple times (by either time or area redundancy) [3], [37]. It turns to a* $[\lambda k, k, \lambda]$*-code where* λ *denotes the number of executions of the algorithm (e.g.,* $\lambda = 2$ *for duplication). The error-detecting capability* $u = \lambda - 1$ *can be straightforwardly improved by increasing* λ *at the cost of multiplying the overhead by* λ*.*

It has also been proposed to use non-linear codes to improve the fault coverage [38], [39]. However, they may have no benefits over linear codes in some scenarios [36]. Nevertheless, many of the issues discussed in this paper are based on the linear property of the underlying code. Therefore, we particularly omit non-linear codes in our constructions.

3 Concept

The effectiveness of faulty-detection property of CED schemes heavily relies on the specific parameters of the underlying code. While the rank directly affect the size of the code $|\mathbf{C}|$, the distance determines the higher bound for the Hamming weight of the detectable error vectors. Many proposed CED schemes are evaluated either by practical experiments limited by the capabilities of the evaluator [40] or only theoretically in the common uniform fault model [32]. In such a model, the error vector $e \in \mathbb{F}_2^n/\{\mathbf{0}\}$ follows a uniform distribution, i.e., $Pr(e) = \frac{1}{2^n-1}$.

Example 3 (Fault Coverage in the Uniform Model)**.** *For an arbitrary* $[n, k, d]$*-code* $\mathbf{C}$*, an error vector* e *cannot be detected by the CED iff* $e \in \mathbf{C}$*. This translate to the following fault coverage considering a uniform fault model, so-called* $\mathcal{U}$*.*

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

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 the fault coverage of* $Cov_{\mathcal{U}}(\mathbf{C}_1) = Cov_{\mathcal{U}}(\mathbf{C}_2) = 0.94$*.*

However, most fault distributions in practice, required by certain fault attacks, should have a specific bias [27],

1. The predictors can have a high complexity, which diminishes the overhead advantage [36].

[28], [41]. Furthermore, the fault coverage can be drastically reduced if the fault distribution changes, e.g., attacking a duplication CED scheme by injecting a **symmetric fault**, i.e., the same fault is injected into A and A′ when A′ = A. Hence, the distance of the code becomes an important factor for the effective fault coverage, e.g., when the adversary has highly-accurate fault-injection facilities (e.g., a laser beam [42], [43]).

It is noteworthy that in some hybrid schemes when both encryption and decryption functionalities are supported by the circuit, the correctness of the computation can be examined by checking whether the decrypted ciphertext matches the original plaintext, or even in a round-based fashion [44]. This for sure increases the fault detection capability, but certain symmetric faults still cannot be detected, e.g., the same fault injected at the input of round 9 of AES encryption and at the output of round 1 of AES decryption.

The stronger an adversary is assumed to be, the more care needs to be taken when implementing a specific CED scheme. In the following, we first introduce our adversary model. 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 in practice. We try to formalize the problem and provide a guideline how CED schemes should be integrated into an implementation of cryptographic algorithms.

3.1 Adversary Model

We assume an adversary model similar to [15], 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. We assume that if a wire is faulty, all its connections are also faulty. In other words, we exclude the cases where the attacker is able to cut a connection and make certain wire(s) faulty without affecting the other wires of the same connection. Since each wire is the output of a cell (either a gate or a register), we can model every fault on a wire as a fault on the corresponding cell.

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

Definition 7 (Multivariate 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 cycles.*

We first focus on univariate $\mathcal{M}_t$ 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 multivariate $\mathcal{M}_t^*$-secure variant.

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

Figure 2 depicts the concepts of checkpoints for an (a) univariate and (b) multivariate adversary with $t = 2$ (the red cells and wires indicate the faulty ones). While the univariate adversary can make at most two cells faulty in one specific clock cycle (gates no. 1 and 3), the multivariate adversary is able to hit at most two cells per clock cycle leading to all output wires 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).

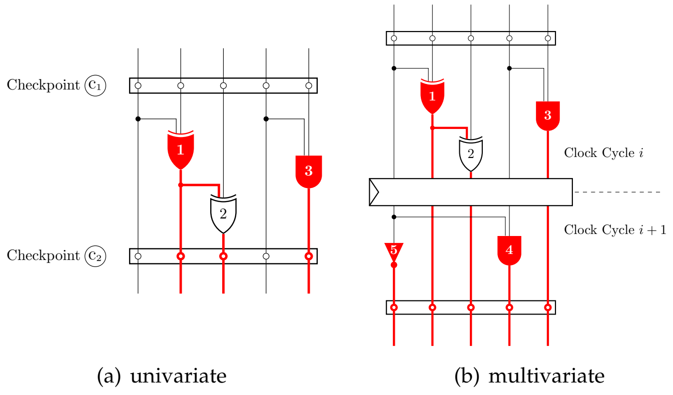


Fig. 2. Fault injection by an $\mathcal{M}_{t=2}$ and $\mathcal{M}^*_{t=2}$ adversary.

In order to conduct a successful DFA, the faults must fit into a particular model, e.g., one-bit fault in a byte, or a single-byte fault in a 16-byte state. For precise models (e.g., single-bit faults) the attack needs a few faulty outputs to recover the secret (for example 2 faulty ciphertexts in case of AES encryption). In contrast, for more general models the number of required faulty outputs increases rapidly [45]. The goal of EDC-based CED schemes is to set a relatively high bound for t, i.e., the attacker has to make more than t cells faulty to obtain a faulty output, hence hardening the corresponding DFA attack.

3.2 Fault Propagation

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 and the other cells involved in the circuit. We formally define fault propagation as follows while it has been briefly discussed in [11], [13], [33].

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 circuit.

This has serious implications as the distance of the underlying code does not provide a reliable bound for t anymore. In particular, this bound is valid for an adversary who can only make faults at the check points. Against an $\mathcal{M}_t$ adversary, however, who can arbitrarily make cells faulty, the distance of the code does not define a meaningful bound.

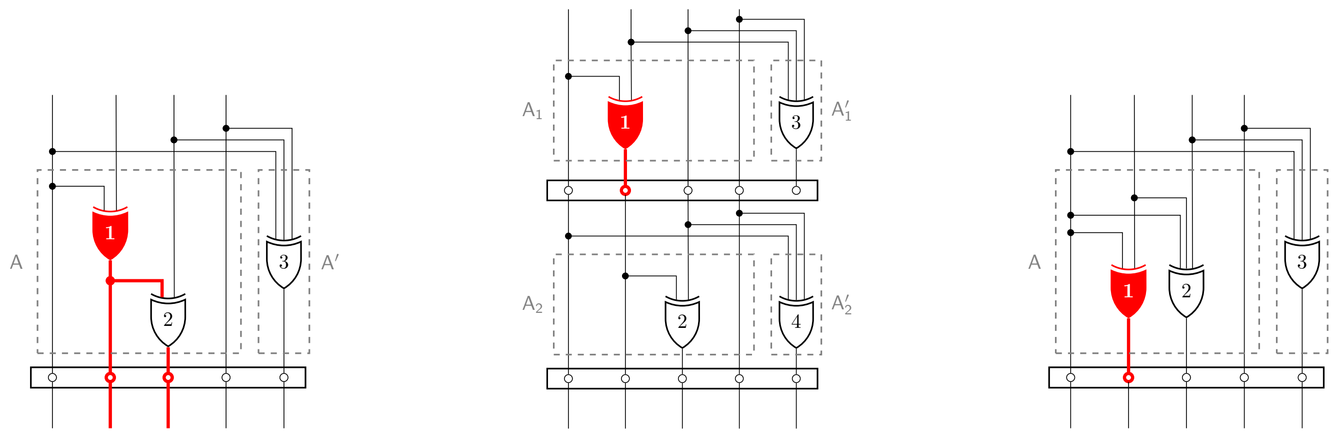


Fig. 3. Realization of T protected with the parity $[5,4,2]$-code. (a) undetectable fault with $t=1$ faulty cells (red), (b) all $t=1$ faulty cells detectable with an **extra checkpoint**, (c) all $t=1$ faulty cells detectable with **forced independence**.

Example 4 (Parity with Fault Propagation)**.** *Let assume a* $[5,4,2]$*-code* $\mathbf{C}_{parity}$ *with a generator matrix*

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

Let 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 protected circuit, including A *and* A′*, is depicted in Figure 3(a) consisting of three 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, based on Equation* (1) *its fault coverage is derived as*

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

Since the code has a distance of $d = 2$*, we restrict the adversary to* $t = d - 1 = 1$ *cells. The code is indeed able to detect all possible faults injected at the gates whose output is exclusively connected to the checkpoints (e.g., Exclusive OR (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 an undetectable faulty state. One example is shown in Figure 3(a), where only one fault is injected on the XOR gate 1. It propagates through the XOR gate 2, and two faulty signals arrive at the checkpoint, i.e., a state undetectable by parity. Repeating $\mathcal{M}_{t=1}$*-bounded faults on all gates of the circuit results in*

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

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 impact of fault propagation is the inclusion of extra checkpoints in the circuit. 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. This concept can be seen as the inclusion of registers in TIs of composed functions to prevent the propagation of glitches [14].

An interesting question is at which points in a circuit the extra checkpoints need to be inserted. An approach is to decompose $T(.)$ 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 limits the fault propagation if each of the sub-functions is less sensitive to fault propagation. To measure the sensitivity, it can be seen that the fault cannot propagate 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.

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. 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 5 (Parity with Extra Checkpoints)**.** *As a generic strategy, adding checkpoints can result in extra combinatorial logic as shown in Figure 3(b) for our previous parity example. The inclusion of an extra checkpoint 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 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

As an observation, duplication (or generally the realization of the target function as λ instantiations of A) provides security against an $\mathcal{M}_{t=\lambda-1}$-bounded adversary even with fault propagation.

Independence Property. The above observation can be generically applied to other codes as well. 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 $T^i : \mathbb{F}_2^k \mapsto \mathbb{F}_2$ in such a way that $\forall i,\ T^i(x) = y^i$. Such a set of component-circuits are called **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 $T^i(.)$.

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. Hence, $t_p = t$ if a checkpoint is placed at the output of $T(.)$.*

Proof. Based on the assumption that the component-circuits 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 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 is shown in Figure 4. Since the hardware synthesizers usually optimize the given design (e.g., by sharing the identical components to achieve lower area), particular attention should be paid to avoid such optimizations[2]. Otherwise, the independence property might be violated.

Example 6 (Parity with Forced Independence)**.** *For our running example, 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). Compared to the former example, forced independence suffices with only one checkpoint which makes it more efficient than $\widehat{\mathbf{C}}_{parity}$, while providing the same fault coverage of*

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

2. 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.

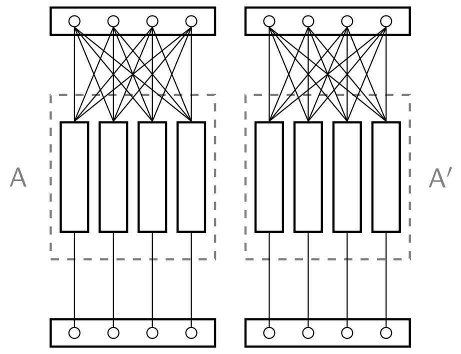


Fig. 4. Forced independence of the target algorithm A and its predictor A′.

4.3 Combination

Each aforementioned solution comes at the cost of higher area specially for complex cryptographic algorithms. Therefore, we propose to utilize a hybrid approach 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.*

Considering a classical symmetric cipher, each sub-function $T_i(.)$ reflects a fundamental module of the cipher (e.g., substitution, diffusion, and key addition).

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 $\sum_j q_{i,j} = q_i$.*

A basic split in a majority of symmetric ciphers follows the cipher's structure, e.g., each sub-function $T_{i,j}$ represent an S-box on nibbles or MixColumns on words.

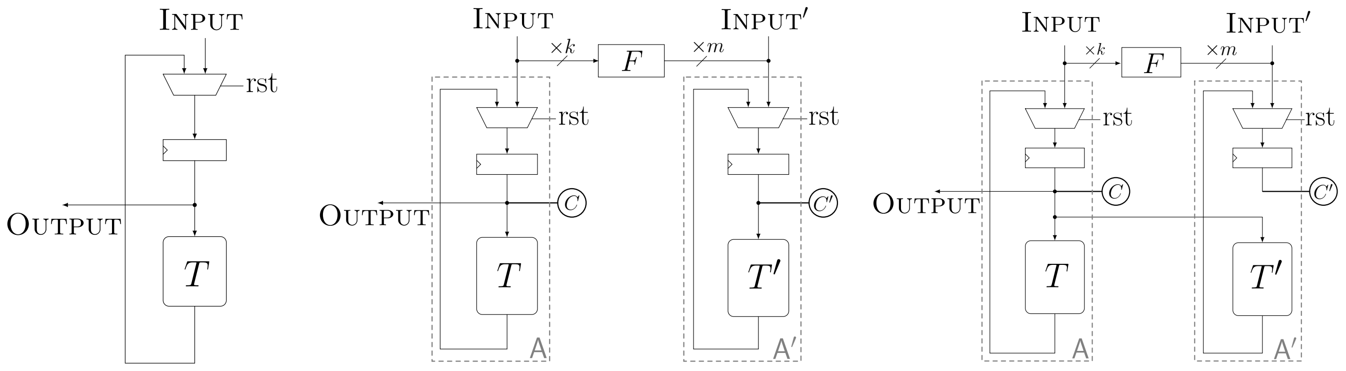
3. *Implement each sub-function $T_{i,j}$ fulfilling the independence property, and place a checkpoint at their output.*

This step benefits from the small input size of $T_{i,j}$. 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, we consider an exemplary algorithm realized by a 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[3].

3. Finite State Machine (FSM) is not shown.



(a) original (b) $F(.)$ injective or transparent (c) $F(.)$ non-injective and non-transparent

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

For the sake of simplicity we 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 use k and m (or $\times k$ and $\times m$) to refer to the bit-length of the message and redundancy, respectively. In the following we distinguish two different cases and explain how the underlying EDC is applied.

F: an injective function or transparent to T

If the redundancy size is at least as large as the message size ($n \geq 2k$) and the function $F(.)$ is injective, 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 $T' = F \circ T \circ F^{-1}$. Both $T(.)$ and $T'(.)$ should be implemented following the forced independence, and a checkpoint is placed at the input of the $T(.)$ marked by Ⓒ and Ⓒ′ in Figure 5(b). It is indeed essential to place the checkpoint at the input of any function implemented by the forced independence. Otherwise, if the checkpoints are moved to the output of $T(.)$, the faults injected at the register cells would potentially propagate to multiple output bits of $T(.)$. 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 independence property. Therefore, any fault injected at $T(.)$ fitting to $\mathcal{M}_{t=d-1}$, is detected at the checkpoint in the next clock cycle.

If $F(.)$ is not injective, for some particular functions $T(.)$ it is still possible for $T'(.)$ to operate only on x' (Figure 5(b)). Since any intermediate value of the circuit should be a valid codeword, if $x' = F(x)$, the output $\langle T(x), T'(x') \rangle$ should also form a valid codeword. This implies that $T'(x') = F(T(x))$, i.e.,

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

Given $T(.)$ 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 is a primitive element (then $a \neq \{0, 1\}$), there exists no such a function $T'(.)$ fitting to Equation (2).

Lemma 4. *Let $T : \mathbb{F}_{2^k} \mapsto \mathbb{F}_{2^k}$ that $T(x) = a \bullet x$ is multiplication with primitive element $a \in \mathbb{F}_{2^k}$, and let $F : \mathbb{F}_{2^k} \mapsto \mathbb{F}_{2^m}$ be any linear non-injective function. 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 words, there are $2^c - 1 \geq 1$ nonzero roots for $F(.)$. Let u be one of them, i.e., $F(u \neq 0) = 0$. Now assume that there exists a $T'(.)$ function which $F \circ T = T' \circ F$. As both $F(.)$ and $T(.)$ are linear, $T'(.)$ is also linear. 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 of $F(.)$. Since $T(.)$ is a multiplication with primitive element 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. □

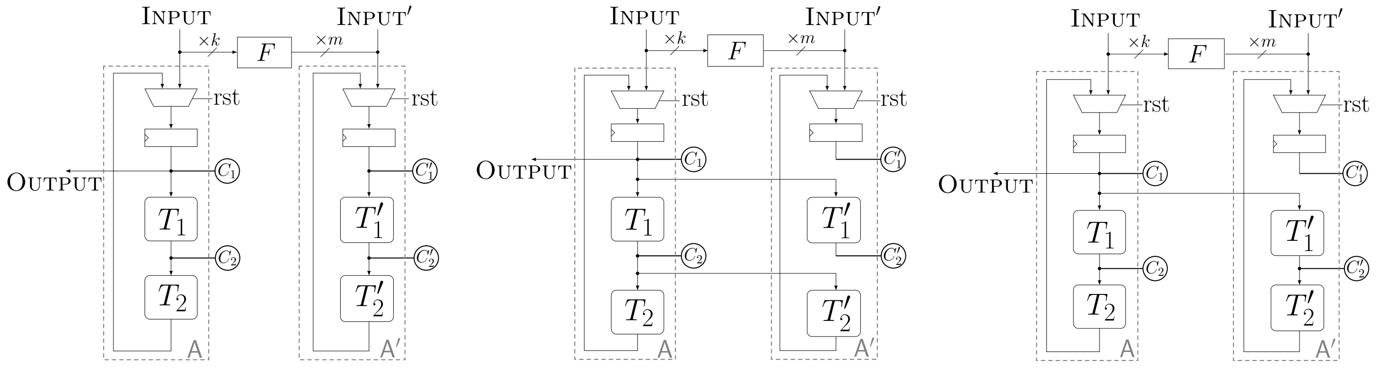
If $T(.)$ is a multiplication with $a = 0/1$ in $\mathbb{F}_{2^k}$, the redundant counterpart $T'(.)$ is a multiplication with the same constant in $\mathbb{F}_{2^m}$. This for example holds for the linear layer of several block ciphers including Midori [6], Skinny and Mantis [5].

F: a non-injective function and non-transparent to T

In such cases, $T'(.)$ needs to receive the original data x 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 is how the $T'(.)$ is realized. It is noteworthy that the implementation of $T'(.)$ should also fulfill the independence property.

Optimization

The decomposition of $T = T_2 \circ T_1$ would generally lead to one of the constructions shown in Figure 6, where a checkpoint is placed between the sub-functions.



(a) $F(.)$ injective or transparent (b) $F(.)$ non-injective and non-transparent (c) $F(.)$ non-injective but transparent to $T_2(.)$

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

Theorem 1. *If $T_2(.)$ is a linear function represented 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 0/1.*

Proof. Below, we represent an intermediate value of the circuit by $\boldsymbol{x}$ (resp. $\boldsymbol{x}'$) as s equally-sized k-bit chunks $\langle x_1, \dots, x_s\rangle$ (resp. m-bit chunks $\langle x'_1, \dots, x'_s\rangle$). Due to the independence property, any fault injected at t cells of a sub-function is modeled by an additive error vector $\boldsymbol{e}$ at its output. We use the notation $\boldsymbol{e}_1 = \langle e_{1,1}, \dots, e_{1,s}\rangle$, $\boldsymbol{e}'_1 = \langle e'_{1,1}, \dots, e'_{1,s}\rangle$, $\boldsymbol{e}_2 = \langle e_{2,1}, \dots, e_{2,s}\rangle$ and $\boldsymbol{e}'_2 = \langle e'_{2,1}, \dots, e'_{2,s}\rangle$ 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 the below equality:

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

where $\boldsymbol{x}$ denotes the fault-free output of $T_1(.)$, i.e., the input of $T_2(.)$. Since $\boldsymbol{x}' = F(\boldsymbol{x})$ and $F \circ T_2 = T'_2 \circ F$, the above equation is simplified to

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

Considering an $\mathcal{M}_{t=d-1}$ adversary model, $wt(\boldsymbol{e}_1) + wt(\boldsymbol{e}'_1) + wt(\boldsymbol{e}_2) + wt(\boldsymbol{e}'_2) < d$. In order to detect it, the relation in Equation (3) should be unequal. It means

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

which equally means

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

We define $T_2(.)$ and $T'_2(.)$ as

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

$$L = \begin{pmatrix} L_{1,1} & L_{1,2} & \cdots & L_{1,s} \\ L_{2,1} & L_{2,2} & \cdots & L_{2,s} \\ \vdots & \vdots & \ddots & \vdots \\ L_{s,1} & L_{s,2} & \cdots & L_{s,s}\end{pmatrix}, L' = \begin{pmatrix} L'_{1,1} & L'_{1,2} & \cdots & L'_{1,s} \\ L'_{2,1} & L'_{2,2} & \cdots & L'_{2,s} \\ \vdots & \vdots & \ddots & \vdots \\ L'_{s,1} & L'_{s,2} & \cdots & L'_{s,s}\end{pmatrix},$$

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

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

Using above definitions, Equation (3) can be written as following s equations.

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

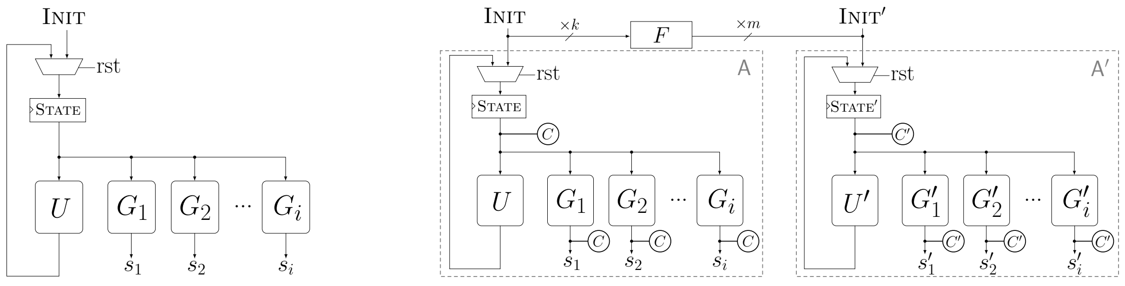
Let us denote $\alpha_i = \bigoplus_{j=1}^{s} e_{1,j} \cdot L_{j,i} \oplus e_{2,i}$ and $\beta_i = \bigoplus_{j=1}^{s} e'_{1,j} \cdot L'_{j,i} \oplus e'_{2,i}$. If $\forall i\ \alpha_i = 0$, then $T_2(\boldsymbol{e}_1) \oplus \boldsymbol{e}_2 = 0$. It means that the output of $T_2(.)$ is fault free,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

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

Without loss of generality, we consider that there exists an i for 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

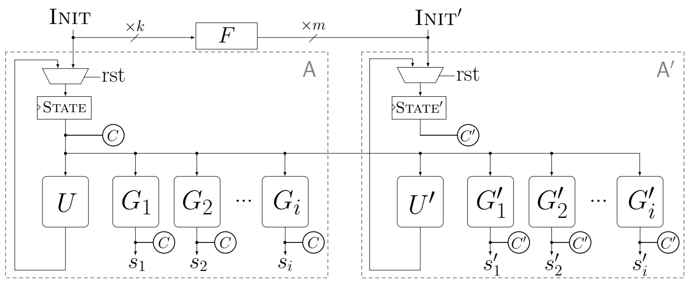
$$\alpha_i \neq 0 \implies wt(\alpha_i) + wt(\alpha_i \cdot P) = wt(\alpha_i) + wt(\beta_i) \geq d. \quad (5)$$

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



(a) original

(b) $F(.)$ injective or transparent



(c) $F(.)$ non-injective and non-transparent

Fig. 7. Application of an EDC on FSM.

Equation (5) as follows.

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

$\square$

This condition holds for MixColumns of several block ciphers including Midori [6], Skinny [5] and Mantis [5].

Control Signals

In contrast to masking countermeasures, the Finite State Machine (FSM) should also be protected against faults. Otherwise, the adversary can change the control flow and obtain faulty results. As a trivial example, by a single-bit fault the attacker may force to terminate an encryption process at the first cipher rounds leading to have 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). An $[n, k, d]$-code can be similarly applied on such a controlling circuit. Each control signal s_i and its redundant counterpart s'_i are related in a form of $s'_i = F(\{0\}^*|s_i)$, i.e., s_i is padded with zero to form a k-bit chunk. In other words, the redundancy of every control signal has a size of m bits. This is essential since $\langle \{0\}^* \mid s, s' \rangle$ should form a valid codeword to guarantee the detection of $t < d$ faulty cells.

- For an injective or transparent $F(.)$, 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 others, the redundancy update function would operate on STATE as $U' = F \circ U$. The same holds for the control signals as $s'_i = G'_i(\text{STATE})$ with $G'_i = F \circ G_i$ (see Figure 7(c)).

Obviously, the implementation of all $U(.)$, $U'(.)$, $G_i(.)$, and $G'_i(.)$ should fulfill the independence property. 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.

Multiplexers

Suppose that s controls a k-bit multiplexer switching between x and y. 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 4.4 formed by a multiplexer tree in m levels. Each row of the multiplexers is controlled by the corresponding bit of s'. The first row by the LSB s'^1, and the last row by MSB s'^m. The m-bit inputs $v_{i\in\{0,\ldots,2^m-1\}}$

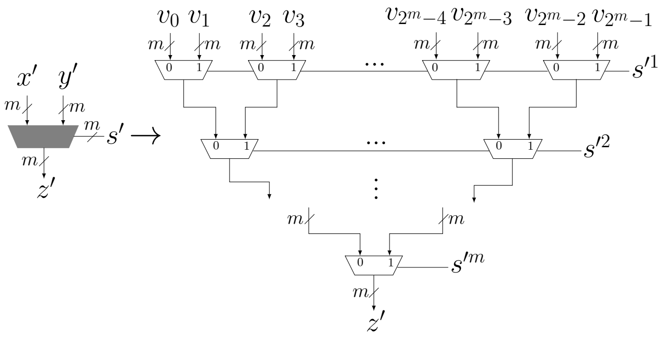


Fig. 8. Application of an EDC on multiplexers.

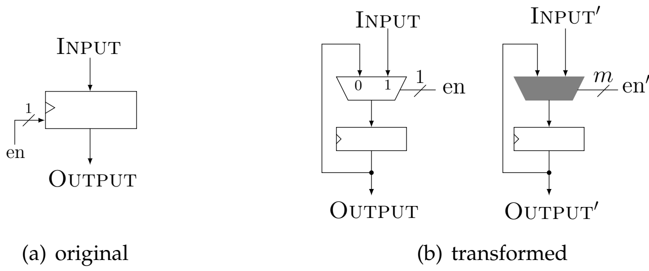


Fig. 9. Application of an EDC on registers with enable.

of the first row are defined as follows:

$$v_i = \begin{cases} x' & ; \quad i = F(0) \\ y' & ; \quad i = F(1) \\ 0 & ; \quad else^4 \end{cases}$$

Since many 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. This does not affect the fault propagation and hence the fault coverage of our construction.

The faults on external signals, those provided through the I/O ports, 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. Therefore, the external control signals (e.g., the rst signal 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 A and A′. This can be seen in Figure 5, Figure 6 and Figure 7.

Registers with Enable
If the circuit contains registers with enable signal, the redundant counterpart cannot trivially make use of the corresponding redundant control signal with bit-length $m > 1$. 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
It is of great importance to integrate a fault detection mechanism into the consistency check process as well. Otherwise, the attacker can target the final module and force a faulty output to pass the consistency check process, independent of the fault coverage of the data-processing part. 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. We propose the construction shown in Figure 10. The values of the checkpoints at the original part $\textcircled{c}_i$ are concatenated[5] 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'. The result of such a check is an m-bit error vector e. To this end, c'' and c' are split into m chunks[6]. The i-th bit of the error vector examines the consistency of all bits of the corresponding i-th chunks:

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

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 4.4). 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 4.4, the input signals of the multiplexer are selected as follows:

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

Finally, the output of the multiplexer is stored in a dedicated register when the computation of the circuit is finished, identified by the 'done' signal. It is among 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'. 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'. To force the faulty OUTPUT to pass the multiplexer, the attacker needs to make at least $d-1$ more cells faulty at the same clock cycle (univariate model) to turn $d-1$ bits vector $\widetilde{e}$ with value fully 1 to fully 0 . This achieves full fault coverage on the entire circuit under the $\mathcal{M}_{t=d-1}$-bounded adversary.

4.5 Extension to Multivariate

As defined in Section 3.1, the $\mathcal{M}_t$ adversary model can make at most t cells faulty at one clock cycle between

4. Arbitrary random values can be given to these inputs v_i without affecting the fault coverage.

5. The single-bit control signals are padded with zero before being concatenated with others.

6. The size of both c' and c'' is always a factor of m bits.

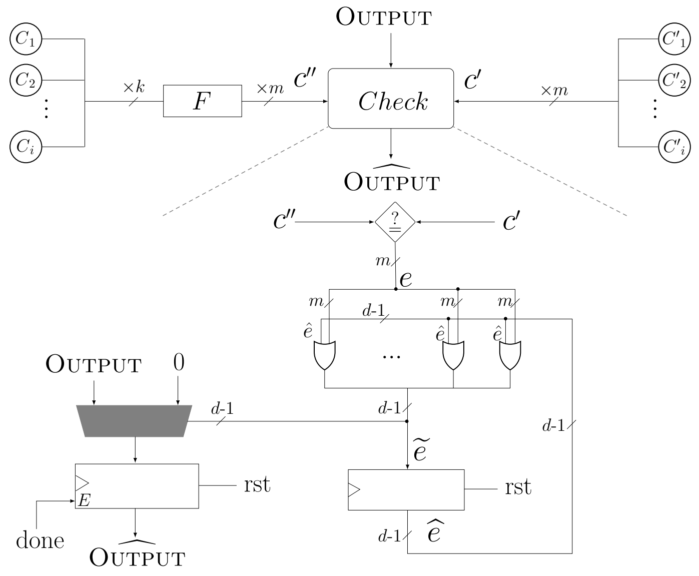


Fig. 10. Application of an EDC on the consistency check ($\mathcal{M}_t$ model).

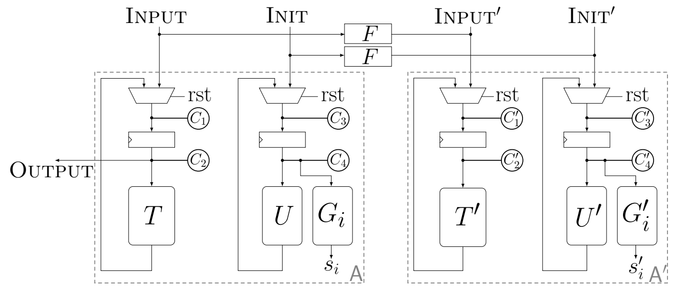


Fig. 11. Supporting the multivariate $\mathcal{M}_t^*$ model.

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 cell in $T(.)$ faulty, which results in a value with a single-bit fault stored in the register. If at the next clock the adversary injects a single-cell 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 cells faulty at every clock cycle, we need to introduce extra checkpoints right at the input of the registers in the design to make sure that the consistency of the values stored in the registers is examined. This includes the register of the data-processing module and that of the FSM (see Figure 11). Hence, in the aforementioned example the fault is detected at the same clock cycle as it is injected. 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. We have to increase the size of the error register $\widehat{e}$ (see Figure 10). Assuming that the adversary injects one fault inside $T(.)$ and $t-1$ faults on the output of the OR operations, it makes only one bit of $\widetilde{e}$ to be '1'. A multivariate adversary would be able to directly set this bit to '0' in the following cycle by targeting the register cell directly and 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 faults per clock cycle. In a general case, suppose that 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, it can be again reduced 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 Boolean masking schemes (e.g., TI [14], [46] and DOM [47]) 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 [13] for an exemplary solution). Note that it is obviously not required to mask the control logic, but all functions (including the masked ones) should fulfill the independence property. As a side note, the combination presented in [13] mixing TI and an EDC is a special case for $m = k = d = 4$. However, it does not deal with fault propagation (i.e., the independence property has been ignored) and the control logic is excluded from the 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 a couple of case studies including PRESENT [4], LED [8], SIMON [9], GIFT [7], Midori [6], Skinny [5] and AES [10]. The analyses and comparisons shown here are based on hardware implementations synthesized by the Synopsys Design Compiler and the IBM 130 nm ASIC standard cell library. By keeping the hierarchy, we made sure that synthesizer does not corrupt the modules designed with independence property.

Among the covered ciphers with 64-bit state, we mainly focus on Skinny. For the rest, we just give their block diagram representations in Appendix.

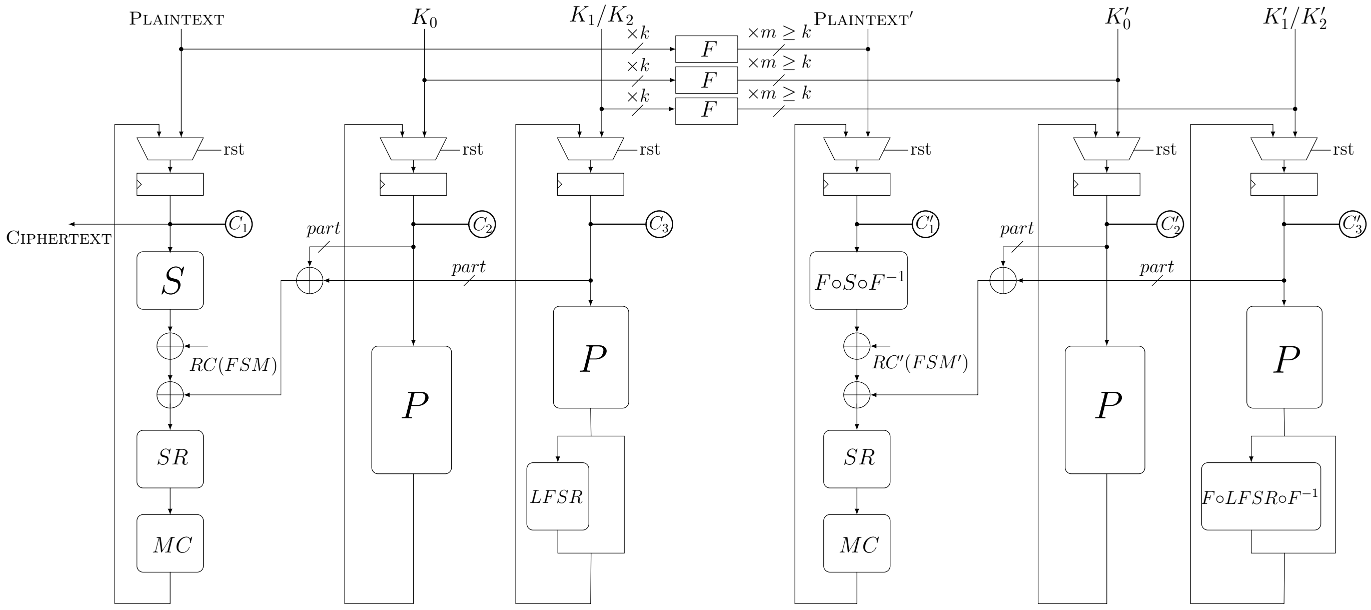


Fig. 12. Skinny, round-based, $F(.)$ injective or transparent

5.1 Skinny-64

The block cipher Skinny-64 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 a 4×4 matrix filled by 0/1. The KeySchedule applies only a nibble-wise permutation P on the entire 64-bit key state.

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 (for 128- and 192-bit key sizes) additionally make use of an LFSR-based operation on each nibble of the first two rows of each 64-bit key after applying the permutation P.

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. 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:

[8,4,4]-code
This implies the case with injective $F(.)$ (see Figure 5(b)). The common code for this case is the extended Hamming-code $[8,4,4]$ (as also used in [13]). The corresponding generator matrix G is

$$G_{eH} = \left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right) = (I_4 \mid P_{eH}),$$

where $F(x) = x \cdot P_{eH}$. 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 0/1 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 12, 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 KeySchedule (shown by P in Figure 12). 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 Linear Feedback Shift Register (LFSR) which is also used as the round counter defined as

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

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 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 at the single control signal 'done'.

[7,4,3]-/ [5,4,2]-code

For these cases, as $m < k$, the $F(.)$ cannot be injective, hence the architecture shown in Figure 5(c) is followed. Based on the following generator matrices, 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.

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

For both cases $F(.)$ is transparent to XORs, ShiftRows, MixColumns, and the KeySchedule permutation, hence can solely operate on STATE$'$. The S-box and the LFSRs are excluded from this list, as it can be seen in Figure 13. The redundant counterpart of the S-box realizes $F \circ S$ and the same holds for the LFSR of the KeySchedule as $F \circ LFSR$.

[2,1,2]-/ [3,1,3]-/ [4,1,4]-code

Duplication, triplication, and quadruplication reflect these codes. 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.The generator matrix of duplication 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 ciphertexts as well as the control signals (see Figure 14 showing a general architecture of such schemes). It is noteworthy that in order to keep full fault coverage, we have used our-proposed consistency check module shown in Section 4.4(§ Checkpoints).

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[7], 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 16 in Appendix). 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 16 to Figure 19 (in Appendix), 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.

Other Ciphers

We have applied our methodology on other symmetric ciphers as well including LED, Midori, PRESENT, GIFT and SIMON. We faced several challenges when the operations do not fit into the nibble/byte-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. The block diagram of their round-based implementations are given in Appendix. 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 '2check' and 'combine' respectively (see Figure 20 and Figure 21 in Appendix). In comparison, the '2check' variant should lead to a smaller area overhead but with higher latency due to its extra checkpoints.

5.2 AES

Due to the popularity of the AES [10], we omit explaining the details of its algorithm. Similar to Skinny, we considered two category of codes, all of which with rank $k = 8$ due to the Sbox size. As an injective function $F(.)$, we focused on $[17, 8, 6]$-, $[19, 8, 7]$-, and $[20, 8, 8]$-code which reflect 9-, 11-, and 12-bit redundancy with a distance of 6, 7, and 8. The P part of the generator matrix $G = [I|P]$ of the selected codes are as follows.

$$P_{17} : \begin{pmatrix} 1&1&1&0&1&0&0&1&0 \\ 1&1&0&1&1&1&0&0&0 \\ 1&0&1&0&1&1&1&0&0 \\ 1&0&0&1&0&1&1&1&0 \\ 0&1&1&0&1&0&1&0&1 \\ 0&1&0&1&0&1&0&1&1 \\ 0&0&1&1&0&0&1&1&1 \\ 0&0&0&0&1&1&1&1&1 \end{pmatrix} \quad P_{19} : \begin{pmatrix} 1&0&1&1&0&0&1&1&0&1&0 \\ 1&0&1&0&1&1&1&0&0&0&1 \\ 1&0&0&1&1&1&0&1&1&0&0 \\ 0&1&1&1&0&1&1&0&1&0&0 \\ 0&1&1&0&1&1&0&1&0&1&0 \\ 0&1&0&1&1&0&1&1&0&0&1 \\ 0&0&1&1&1&0&0&0&1&1&1 \\ 0&0&0&0&0&1&1&1&1&1&1 \end{pmatrix}$$

$$P_{20} : \begin{pmatrix} 1&1&1&1&1&0&1&0&0&1&0&0 \\ 1&1&1&0&0&1&1&1&0&0&1&0 \\ 1&0&1&1&0&1&0&1&1&1&0&0 \\ 1&0&0&1&1&1&1&0&1&0&1&0 \\ 0&1&1&0&1&1&0&1&0&1&0&1 \\ 0&1&0&1&1&0&1&1&0&0&1&1 \\ 0&0&1&1&1&0&0&0&1&1&1&1 \\ 0&0&0&0&0&1&1&1&1&1&1&1 \end{pmatrix}$$

In such cases, while $F(.)$ is transparent to AddRoundKey and ShiftRows, the redundancy part of Sbox, MixColumns and FSM (the round constant) are implemented by $F \circ T \circ F^{-1}$ (see Figure 5(b)). The same holds for the 4 Sboxes used

7. Excluding the bit-serial fashion [48].

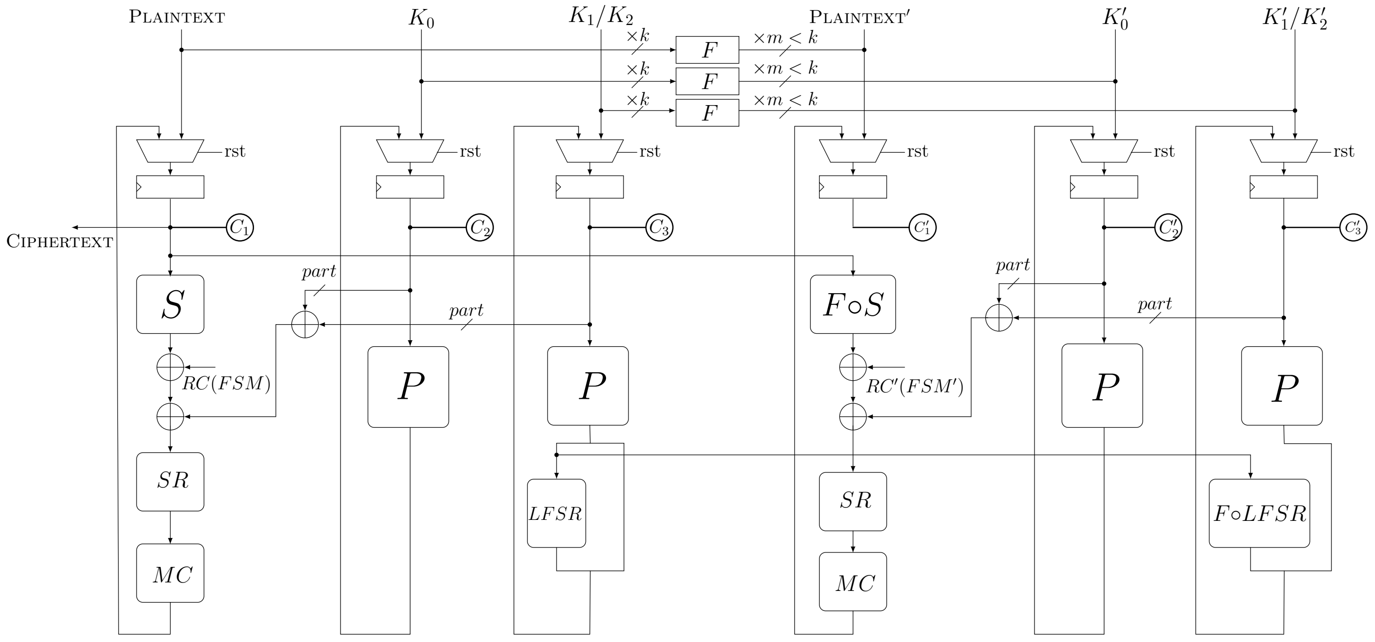


Fig. 13. Skinny, round-based, $F(.)$ non-injective but transparent to SR, MC and P

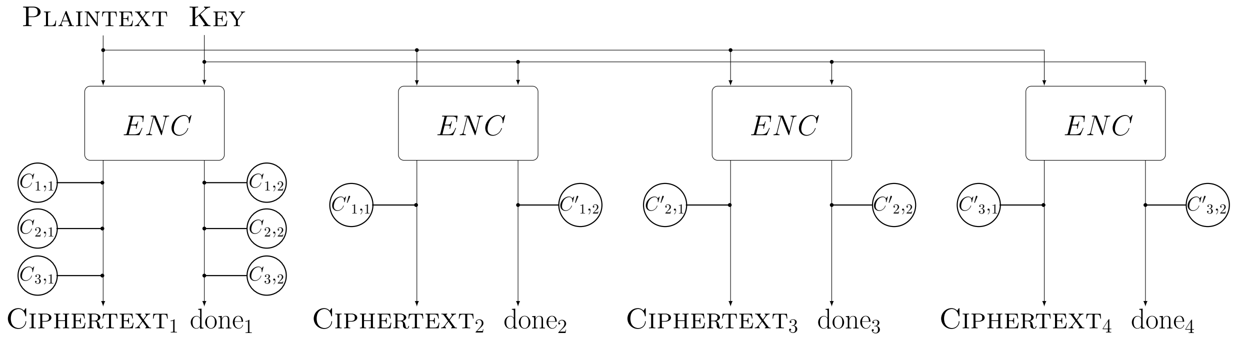


Fig. 14. Duplication, triplication, and quadruplication concept

in the KeySchedule. In addition to the checkpoints placed at the Sbox inputs, extra checkpoints have to be inserted at the MixColumns input since its corresponding matrix is not filled only by 0/1 [10].

For non-injective $F(.)$, we considered $[9, 8, 2]$-, $[12, 8, 3]$-, $[13, 8, 4]$-, and $[16, 8, 5]$-codes. Note that they are smallest codes with rank $k = 8$ and the considered distances 2, 3, 4, and 5. The first code $[9, 8, 2]$ is the common 8-bit parity code, and the P part of the generator matrix of the others are given below.

$$P_{12}: \begin{pmatrix} 1&0&1&1\\ 0&1&1&1\\ 1&1&0&0\\ 1&0&1&0\\ 1&0&0&1\\ 0&1&1&0\\ 0&1&0&1\\ 0&0&1&1 \end{pmatrix} \quad P_{13}: \begin{pmatrix} 1&1&1&0&0\\ 1&1&0&1&0\\ 1&0&1&0&1\\ 1&0&0&1&1\\ 0&1&1&1&0\\ 0&1&1&0&1\\ 0&1&0&1&1\\ 0&0&1&1&1 \end{pmatrix} \quad P_{16}: \begin{pmatrix} 1&1&1&0&0&0&0&1\\ 1&1&0&1&1&0&0&0\\ 1&0&1&0&1&1&0&0\\ 1&0&0&1&0&1&1&0\\ 0&1&1&0&1&0&1&0\\ 0&1&0&1&0&1&0&1\\ 0&0&1&1&0&0&1&1\\ 0&0&0&0&1&1&1&1 \end{pmatrix}$$

The redundant part of XORs and ShiftRows as well as the checkpoints are the same as the ones with injective $F(.)$. The only difference is on the other modules which should receive the original data and compute $F \circ T$ (see Figure 5(c)).

5.3 Comparison

Considering both univariate and multivariate adversary models, we summarize the area and latency figures of our implementations 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[8]) our approach outperforms the duplication schemes. However, such benefits depend on the target algorithm. For instance, in almost all cases of LED with 128-bit key 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.

We should also highlight that we have implemented the 'plain' (unprotected) version of the considered ciphers by ourselves under the same fashion and the same design

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

TABLE 1
Area (GE) and Latency (ns) of our implementations considering different $[n, k, d]$-codes, using IBM 130 nm ASIC library.

Algorithm	Attacker	clock	plain		$[\mathbf{5},\mathbf{4},\mathbf{2}]$		$[\mathbf{7},\mathbf{4},\mathbf{3}]$		$[\mathbf{8},\mathbf{4},\mathbf{4}]$		duplication $[\mathbf{2},\mathbf{1},\mathbf{2}]$		triplication $[\mathbf{3},\mathbf{1},\mathbf{3}]$		quadrupl. $[\mathbf{4},\mathbf{1},\mathbf{4}]$	
	Model	cycles	area	lat.	area	lat.	area	lat.	area	lat.	area	lat.	area	lat.	area	lat.
Skinny-64	$\mathcal{M}_t$	33	1243	4.11	2732	5.27	4153	6.08	5041	6.43	3155	4.53	4649	5.04	6147	5.68
64-bit key	$\mathcal{M}_t^*$				3192	7.91	5085	8.25	6294	8.94						
Skinny-64 ser.	$\mathcal{M}_t$	688	990	4.24	1924	5.53	3383	5.75	4870	7.66	2019	4.28	3028	4.29	4042	4.78
64-bit key	$\mathcal{M}_t^*$				2250	8.71	4115	9.06	5855	11.55						
LED 2check	$\mathcal{M}_t$	33	1665	7.72	3667	9.17	5697	11.52	7226	12.15	3968	7.69	5874	7.71	7784	7.75
64-bit key	$\mathcal{M}_t^*$				3986	12.97	6316	13.41	8086	14.52						
LED combine	$\mathcal{M}_t$				4155	8.45	6039	9.44	7716	10.40						
64-bit key	$\mathcal{M}_t^*$				4508	12.57	6720	13.47	8569	13.47						
Skinny-64	$\mathcal{M}_t$	37	1738	3.66	3640	5.16	5636	6.09	6804	6.37	4128	4.34	6109	4.86	8095	5.50
128-bit key	$\mathcal{M}_t^*$				4236	7.84	6879	8.85	8477	9.60						
Skinny-64 ser.	$\mathcal{M}_t$	772	1446	4.03	2778	5.42	4867	5.92	6883	7.18	2906	5.55	4358	5.55	5815	5.55
128-bit key	$\mathcal{M}_t^*$				3228	9.26	5895	9.94	8257	11.56						
LED 2check	$\mathcal{M}_t$	49	1664	9.15	3996	9.81	6359	11.66	8240	13.40	3991	9.37	5907	9.37	7822	9.36
128-bit key	$\mathcal{M}_t^*$				4320	13.51	6972	13.71	9099	16.18						
LED combine	$\mathcal{M}_t$				4499	9.57	6699	10.04	8718	12.80						
128-bit key	$\mathcal{M}_t^*$				4813	13.11	7359	12.96	9637	15.89						
Midori	$\mathcal{M}_t$	17	1372	7.57	3282	8.25	5262	8.87	6840	10.40	3412	8.81	5029	9.51	6657	9.85
	$\mathcal{M}_t^*$				3615	11.18	5891	11.52	7693	14.30						
PRESENT	$\mathcal{M}_t$	32	1767	2.93	4211	5.19	6639	6.32	8219	6.71	4174	4.59	6179	5.10	8186	5.78
	$\mathcal{M}_t^*$				4792	7.83	7899	8.87	9896	9.37						
GIFT	$\mathcal{M}_t$	29	1587	2.88	3824	5.11	6082	6.11	7767	6.61	3847	4.33	5688	4.83	7533	5.47
	$\mathcal{M}_t^*$				4420	7.49	7325	8.56	9432	9.01						
SIMON	$\mathcal{M}_t$	45	1629	2.86	3614	5.20	5621	5.93	7603	6.44	3912	4.47	5785	4.97	7663	5.61
	$\mathcal{M}_t^*$				4211	7.28	6866	8.03	9277	9.97						
Skinny-64	$\mathcal{M}_t$	41	2206	4.00	4540	5.63	7119	6.34	8553	6.74	5054	4.47	7494	5.00	9940	5.62
192-bit key	$\mathcal{M}_t^*$				5272	7.69	8676	8.79	10640	9.34						
Skinny-64 ser.	$\mathcal{M}_t$	856	1896	5.12	3602	5.28	6321	5.76	8893	7.53	3807	4.44	5710	4.44	7618	4.76
192-bit key	$\mathcal{M}_t^*$				4219	8.55	7698	9.21	10713	11.42						

Algorithm	Attacker	clock	plain		$[\mathbf{9},\mathbf{8},\mathbf{2}]$		$[\mathbf{12},\mathbf{8},\mathbf{3}]$		$[\mathbf{13},\mathbf{8},\mathbf{4}]$		$[\mathbf{16},\mathbf{8},\mathbf{5}]$		$[\mathbf{17},\mathbf{8},\mathbf{6}]$		$[\mathbf{19},\mathbf{8},\mathbf{7}]$		$[\mathbf{20},\mathbf{8},\mathbf{8}]$		$[\mathbf{2},\mathbf{1},\mathbf{2}]$		$[\mathbf{3},\mathbf{1},\mathbf{3}]$		$[\mathbf{4},\mathbf{1},\mathbf{4}]$	
	Model	cycles	area	lat.	area	lat.	area	lat.	area	lat.	area	lat.	area	lat.	area	lat.	area	lat.	area	lat.	area	lat.	area	lat.
AES	$\mathcal{M}_t$	11	13327	7.63	25789	9.78	36679	10.99	40200	12.47	51116	12.84	54370	11.58	62956	12.47	66039	13.27	27988	8.06	41865	7.68	55769	8.10
128-bit key	$\mathcal{M}_t^*$				26691	13.57	38135	14.50	42599	15.01	54285	16.51	58305	16.83	67606	17.75	71411	18.60						

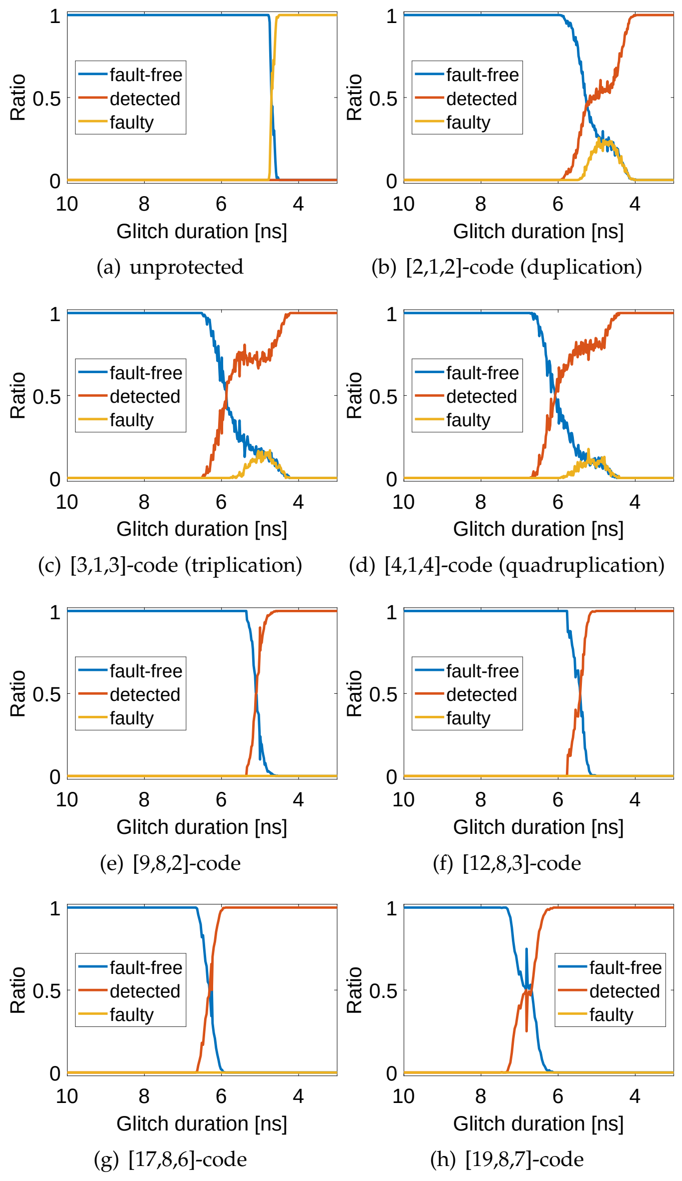
architecture. Hence, the area footprints reported in Table 1 for 'plain' implementations do not necessarily fit to those reported in original articles [5]. For the 'plain' implementations, we further did not fulfill the independence property, did not keep the hierarchy, and did not make use of any special scan-flipflops, employed in several designs to reduce the area footprint (e.g., in [4], [5]).

As given in Section 4.5, to resist against a 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. Since such extra checkpoints are placed at the registers' input, the latency of the circuit is also increased by a roughly constant value.

5.4 Fault Detection

In order to examine the fault-detection ability of our constructions, we conducted both simulation and practical experiments. Injecting faults fitting to our defined models needs ASIC fabrication and laser beams. On the other hand, no fault-diagnosis tools for cryptographic designs is known where the adversary model can be accurately defined. Hence, we have conducted a few custom simulations. For a given design, we have taken the net-list generated during the synthesis process, and replaced every cell with the corresponding one whose output is toggled by a fault signal. This way, we can control every cell of the synthesized circuit including the data-processing, control logic, and check parts. As an example, an implementation of Skinny-64 protected with the [5,4,2]-code against a univariate adversary $\mathcal{M}_{t=1}$ contains 849 cells, i.e., a vector of 849 signals to inject faults. Our simulations under the considered adversary model (i.e., single-bit faults at one clock cycle for every encryption) showed 100% fault coverage. We have repeated this process on our other implementations (with larger codes as well as multivariate adversary model). In all cases, we have not observed any undetected fault fitting into the bounded model.

For the practical experiments, we made use of a Basys 3 toolkit board from Digilent equipped with an Artix-7 FPGA [49]. We followed the principle expressed in [50] to inject faults by clock glitch, i.e., a signal generator provides an external clock signal with adjustable frequency which is multiplied by a constant factor inside the FPGA by a Digital Clock Management (DCM) unit, and the ordinary clock is replaced by the fast clock at selected clock cycles to inject clock glitch. For a given design we have started to decrease the clock glitch duration to get faulty values. This way we have no precise control over the number of injected faults, but by shortening the clock glitch we first violate the delay of the critical path of the circuit. For each clock glitch period, we gave the circuit 1000 random plaintexts (with a fixed key), and extracted the ratios of fault-free, detected, and faulty results. Figure 15 demonstrates such ratios over the duration of the clock glitch for different AES designs including unprotected, duplication/triplication/quadruplication, and the ones protected by our scheme using various codes. In all cases we evaluated the variants protected against the univariate $\mathcal{M}_t$ model, and injected the clock glitch at the last encryption round. The benefit of our constructions compared to the unprotected design is clearly shown, but compared to duplication/triplication/quadruplication it can be seen that there are some faults which pass through the duplication mechanisms (i.e., symmetric faults) which are fully avoided in our designs. Since the duplicated modules A and A′ are implemented similarly, they have very similar critical path; hence a clock glitch has some times equal effect on both/multiple instances. This does not happen in our constructions leading to 0 faulty ratio.



(a) unprotected
(b) [2,1,2]-code (duplication)
(c) [3,1,3]-code (triplication)
(d) [4,1,4]-code (quadruplication)
(e) [9,8,2]-code
(f) [12,8,3]-code
(g) [17,8,6]-code
(h) [19,8,7]-code

Fig. 15. Result of fault injection by clock glitch on different implementations of AES.

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.

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. We made the HDL code of our entire designs publicly available through https://github.com/emsec/ImpeccableCircuits.

Regarding future works, an ASIC-based practical evaluation of the fault-resistance of our designs using laser fault injections 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) [51]. 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.

References

[1] D. Boneh, R. A. DeMillo, and R. J. Lipton, “On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract),” in *EUROCRYPT*, ser. LNCS, vol. 1233. Springer, 1997, pp. 37–51.

[2] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, “The Sorcerer’s Apprentice Guide to Fault Attacks,” *Proceedings of the IEEE*, vol. 94, no. 2, pp. 370–382, 2006.

[3] X. Guo, D. Mukhopadhyay, C. Jin, and R. Karri, “Security analysis of concurrent error detection against differential fault analysis,” *J. Cryptographic Eng.*, vol. 5, no. 3, pp. 153–169, 2015.

[4] 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*, ser. LNCS, vol. 4727. Springer, 2007, pp. 450–466.

[5] 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*, ser. LNCS, vol. 9815. Springer, 2016, pp. 123–153.

[6] S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, and F. Regazzoni, “Midori: A Block Cipher for Low Energy,” in *ASIACRYPT*, ser. LNCS, vol. 9453. Springer, 2015, pp. 411–436.

[7] 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*, ser. LNCS, vol. 10529. Springer, 2017, pp. 321–345.

[8] J. Guo, T. Peyrin, A. Poschmann, and M. J. B. Robshaw, “The LED Block Cipher,” in *CHES*, ser. LNCS, vol. 6917. Springer, 2011, pp. 326–341.

[9] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, “The SIMON and SPECK lightweight block ciphers,” in *DAC*. ACM, 2015, pp. 175:1–175:6.

[10] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*, ser. Information Security and Cryptography. Springer, 2002.

[11] 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*, 2005, pp. 141–146.

[12] 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*, vol. 7, no. 1, pp. 75–85, 2017.

[13] T. Schneider, A. Moradi, and T. Güneysu, “ParTI - Towards Combined Hardware Countermeasures Against Side-Channel and Fault-Injection Attacks,” in *CRYPTO*, ser. LNCS, vol. 9815. Springer, 2016, pp. 302–332.

[14] S. Nikova, V. Rijmen, and M. Schläffer, “Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches,” *J. Cryptology*, vol. 24, no. 2, pp. 292–321, 2011.

[15] Y. Ishai, M. Prabhakaran, A. Sahai, and D. Wagner, “Private Circuits II: Keeping Secrets in Tamperable Circuits,” in *EUROCRYPT*, ser. LNCS, vol. 4004, 2006, pp. 308–327.

[16] T. D. Cnudde and S. Nikova, “More Efficient Private Circuits II through Threshold Implementations,” in *FDTC*. IEEE Computer Society, 2016, pp. 114–124.

[17] O. Seker, A. Fernandez-Rubio, T. Eisenbarth, and R. Steinwandt, “Extending glitch-free multiparty protocols to resist fault injection attacks,” *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, no. 3, pp. 394–430, 2018.

[18] O. Reparaz, L. De Meyer, B. Bilgin, V. Arribas, S. Nikova, V. Nikov, and N. P. Smart, “CAPA: the spirit of beaver against physical attacks,” in *CRYPTO 2018*, ser. LNCS, vol. 10991. Springer, 2018, pp. 121–151.

[19] B. Yuce, N. F. Ghalaty, and P. Schaumont, “TVVF: Estimating the vulnerability of hardware cryptosystems against timing violation attacks,” in *HOST*. IEEE, 2015, pp. 72–77.

[20] N. Selmane, S. Guilley, and J. Danger, “Practical Setup Time Violation Attacks on AES,” in *EDCC-7*, 2008, pp. 91–96.

[21] M. Agoyan, J. Dutertre, D. Naccache, B. Robisson, and A. Tria, “When Clocks Fail: On Critical Paths and Clock Faults,” in *CARDIS*, ser. LNCS, vol. 6035, 2010, pp. 182–193.

[22] J.-J. Quisquater and D. Samyde, “Eddy current for magnetic analysis with active sensor,” in *Proceedings of Esmart*, 2002.

[23] S. P. Skorobogatov and R. J. Anderson, “Optical Fault Induction Attacks,” in *CHES*, ser. LNCS, vol. 2523. Springer, 2002, pp. 2–12.

[24] B. Selmke, J. Heyszl, and G. Sigl, “Attack on a DFA Protected AES by Simultaneous Laser Fault Injections,” in *FDTC*. IEEE Computer Society, 2016, pp. 36–46.

[25] 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*, ser. LNCS, vol. 8419. Springer, 2013, pp. 236–252.

[26] 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*. IEEE Computer Society, 2015, pp. 14–27.

[27] N. F. Ghalaty, B. Yuce, M. M. I. Taha, and P. Schaumont, “Differential Fault Intensity Analysis,” in *FDTC*. IEEE Computer Society, 2014, pp. 49–58.

[28] T. Fuhr, É. Jaulmes, V. Lomné, and A. Thillard, “Fault Attacks on AES with Faulty Ciphertexts Only,” in *FDTC*. IEEE Computer Society, 2013, pp. 108–118.

[29] E. Biham and A. Shamir, “Differential Fault Analysis of Secret Key Cryptosystems,” in *CRYPTO*, ser. LNCS, vol. 1294. Springer, 1997, pp. 513–525.

[30] 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*, ser. LNCS, vol. 6917. Springer, 2011, pp. 292–311.

[31] C. Dobraunig, M. Eichlseder, T. Korak, S. Mangard, F. Mendel, and R. Primas, “SIFA: exploiting ineffective fault inductions on symmetric cryptography,” *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, no. 3, pp. 547–572, 2018.

[32] 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*, vol. 52, no. 4, pp. 492–505, 2003.

[33] 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*. IEEE, 2016, pp. 445–450.

[34] C. Yen and B. Wu, “Simple Error Detection Methods for Hardware Implementation of Advanced Encryption Standard,” *IEEE Trans. Computers*, vol. 55, no. 6, pp. 720–731, 2006.

[35] F. J. MacWilliams and N. J. A. Sloane, *The theory of error correcting codes*, ser. North-Holland mathematical library. North-Holland Pub. Co. New York, 1977.

[36] T. Malkin, F. Standaert, and M. Yung, “A Comparative Cost/Security Analysis of Fault Attack Countermeasures,” in *FDTC*, ser. LNCS, vol. 4236. Springer, 2006, pp. 159–172.

[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*, vol. 21, no. 12, pp. 1509–1517, 2002.

[38] K. D. Akdemir, Z. Wang, M. G. Karpovsky, and B. Sunar, “Design of Cryptographic Devices Resilient to Fault Injection Attacks Us-

ing Nonlinear Robust Codes," in *Fault Analysis in Cryptography*. Springer, 2012, pp. 171–199.

[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*. IEEE Computer Society, 2008, pp. 41–50.

[40] S. Patranabis, A. Chakraborty, P. H. Nguyen, and D. Mukhopadhyay, "A Biased Fault Attack on the Time Redundancy Countermeasure for AES," in *COSADE*, ser. LNCS, vol. 9064. Springer, 2015, pp. 189–203.

[41] S. Guilley, L. Sauvage, J. Danger, N. Selmane, and R. Pacalet, "Silicon-level Solutions to Counteract Passive and Active Attacks," in *FDTC*. IEEE, 2008, pp. 3–17.

[42] M. Agoyan, J. Dutertre, A. Mirbaha, D. Naccache, A. Ribotta, and A. Tria, "How to flip a bit?" in *IOLTS*. IEEE Computer Society, 2010, pp. 235–239.

[43] F. Courbon, P. Loubet-Moundi, J. J. A. Fournier, and A. Tria, "Adjusting Laser Injections for Fully Controlled Faults," in *COSADE*, ser. LNCS, vol. 8622, 2014, pp. 229–242.

[44] A. Satoh, T. Sugawara, N. Homma, and T. Aoki, "High-Performance Concurrent Error Detection Scheme for AES Hardware," in *CHES 2008*, ser. LNCS, vol. 5154. Springer, 2008, pp. 100–112.

[45] A. Moradi, M. T. M. Shalmani, and M. Salmasizadeh, "A Generalized Method of Differential Fault Attack Against AES Cryptosystem," in *CHES 2006*, ser. LNCS, vol. 4249. Springer, 2006, pp. 91–100.

[46] O. Reparaz, B. Bilgin, S. Nikova, B. Gierlichs, and I. Verbauwhede, "Consolidating Masking Schemes," in *CRYPTO 2015*, ser. LNCS, vol. 9215. Springer, 2015, pp. 764–783.

[47] H. Groß, S. Mangard, and T. Korak, "An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order," in *CT-RSA 2017*, ser. LNCS, vol. 10159. Springer, 2017, pp. 95–112.

[48] 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*, ser. LNCS, vol. 10529. Springer, 2017, pp. 687–707.

[49] Digilent, "Basys3," 2019, https://reference.digilentinc.com/reference/programmable-logic/basys-3/.

[50] A. Moradi, O. Mischke, and C. Paar, "One Attack to Rule Them All: Collision Timing Attack versus 42 AES ASIC Cores," *IEEE Trans. Computers*, vol. 62, no. 9, pp. 1786–1798, 2013.

[51] C. Dobraunig, M. Eichlseder, H. Groß, S. Mangard, F. Mendel, and R. Primas, "Statistical Ineffective Fault Attacks on Masked AES with Fault Countermeasures," in *ASIACRYPT 2018*, ser. LNCS, vol. 11273. Springer, 2018, pp. 315–342.

APPENDIX

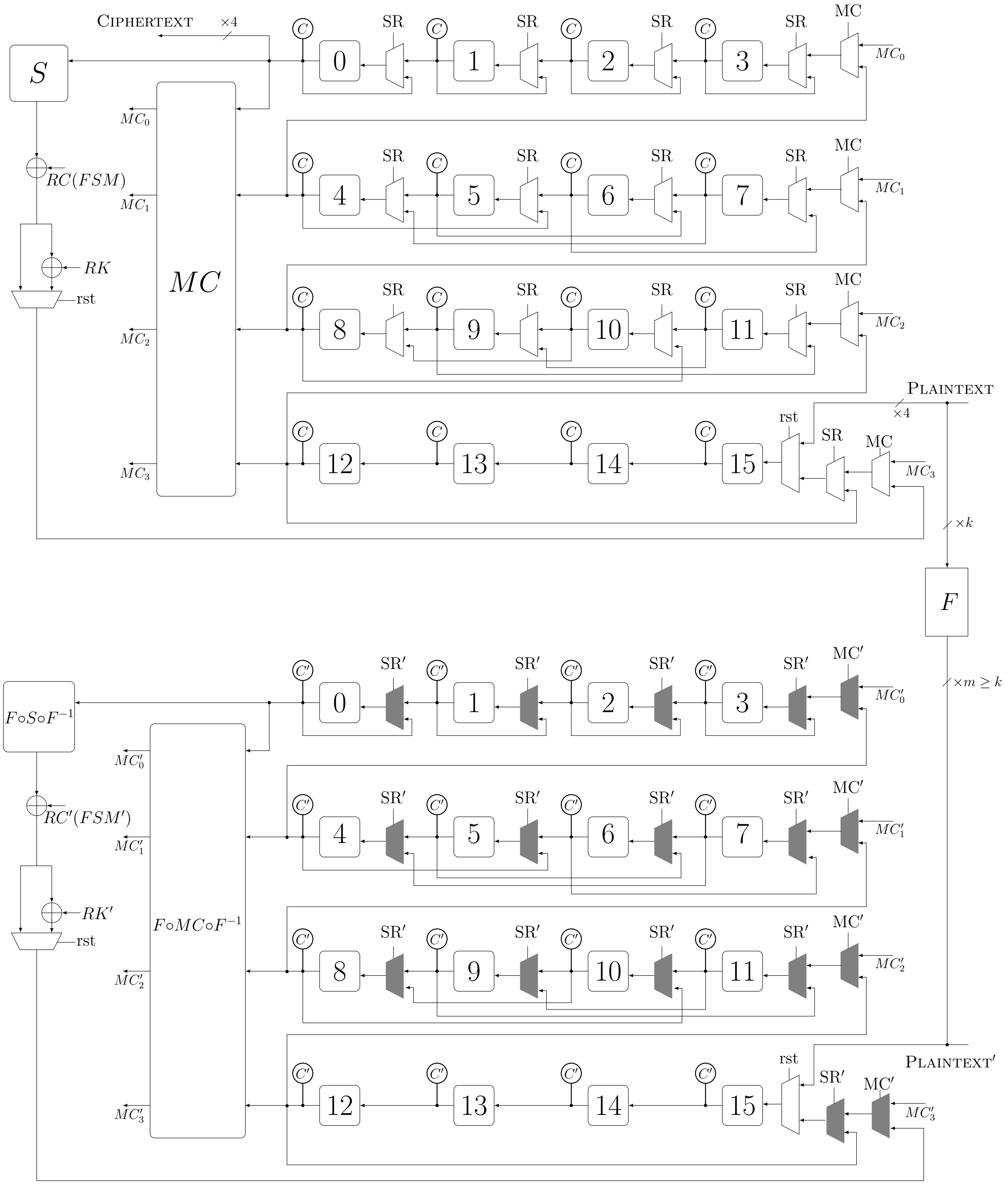


Fig. 16. Skinny, nibble-serial, data path, $m \geq k$.

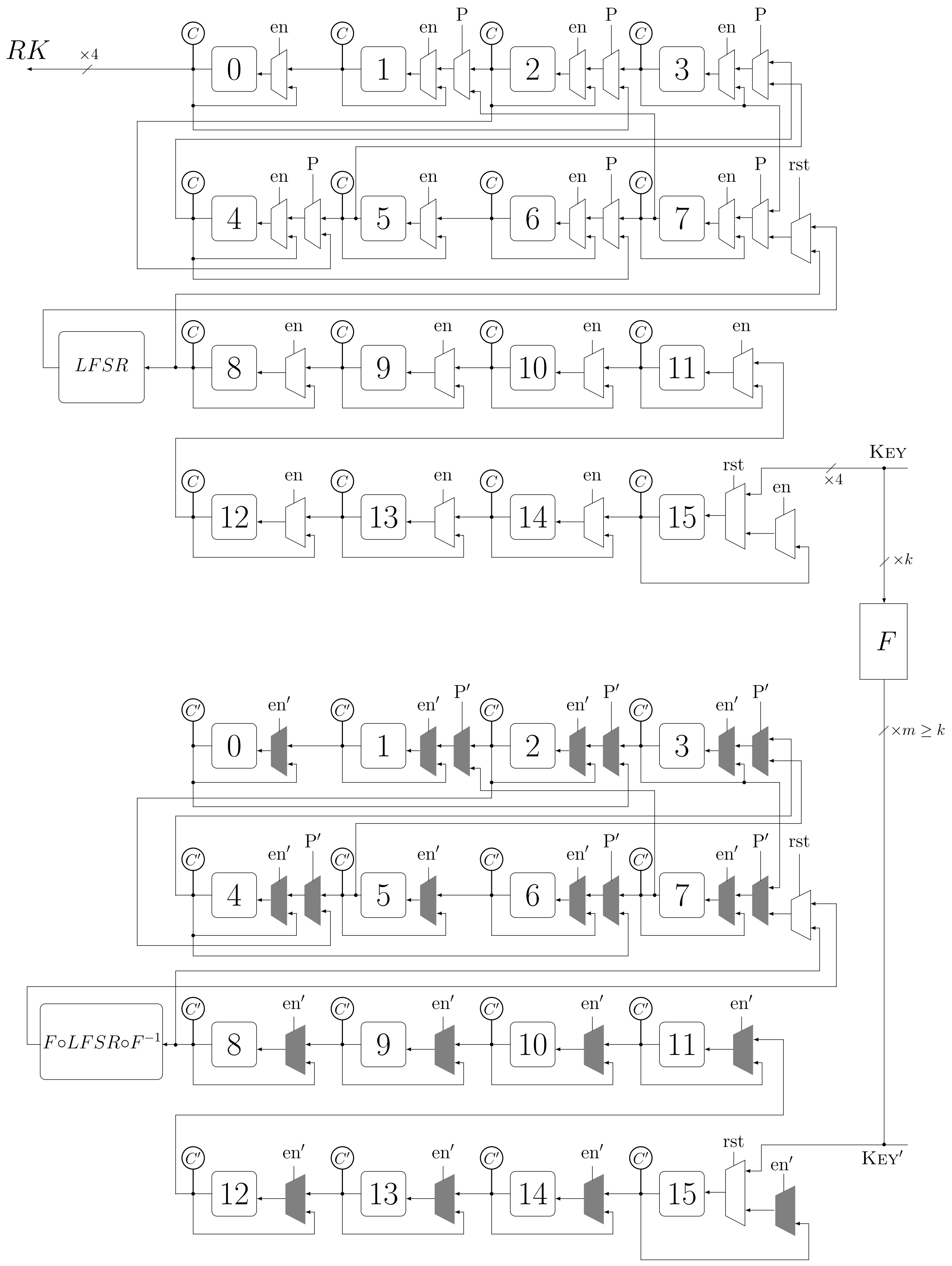


Fig. 17. Skinny, nibble-serial, key path, $m \geq k$.

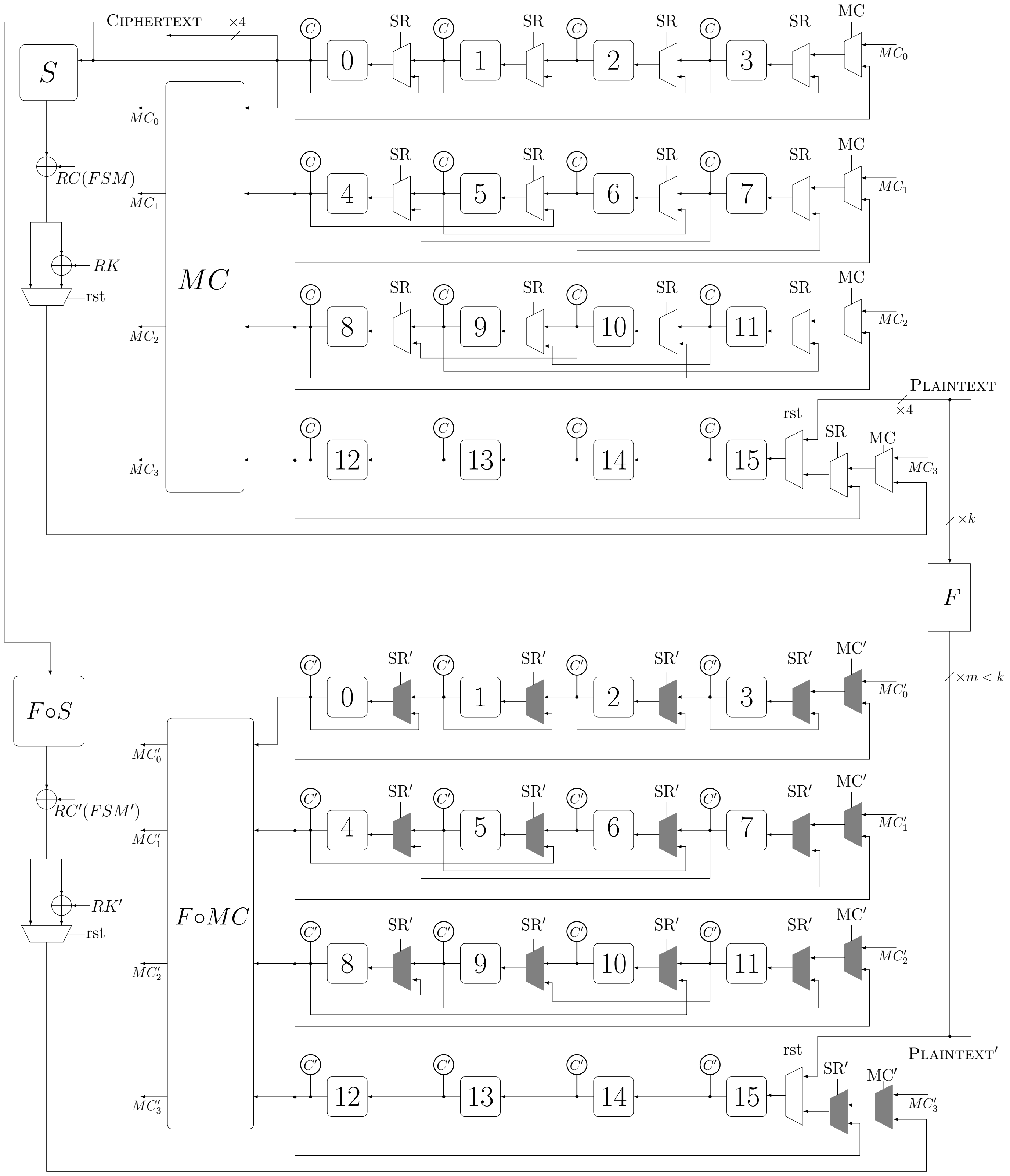


Fig. 18. Skinny, nibble-serial, data path, $m < k$.

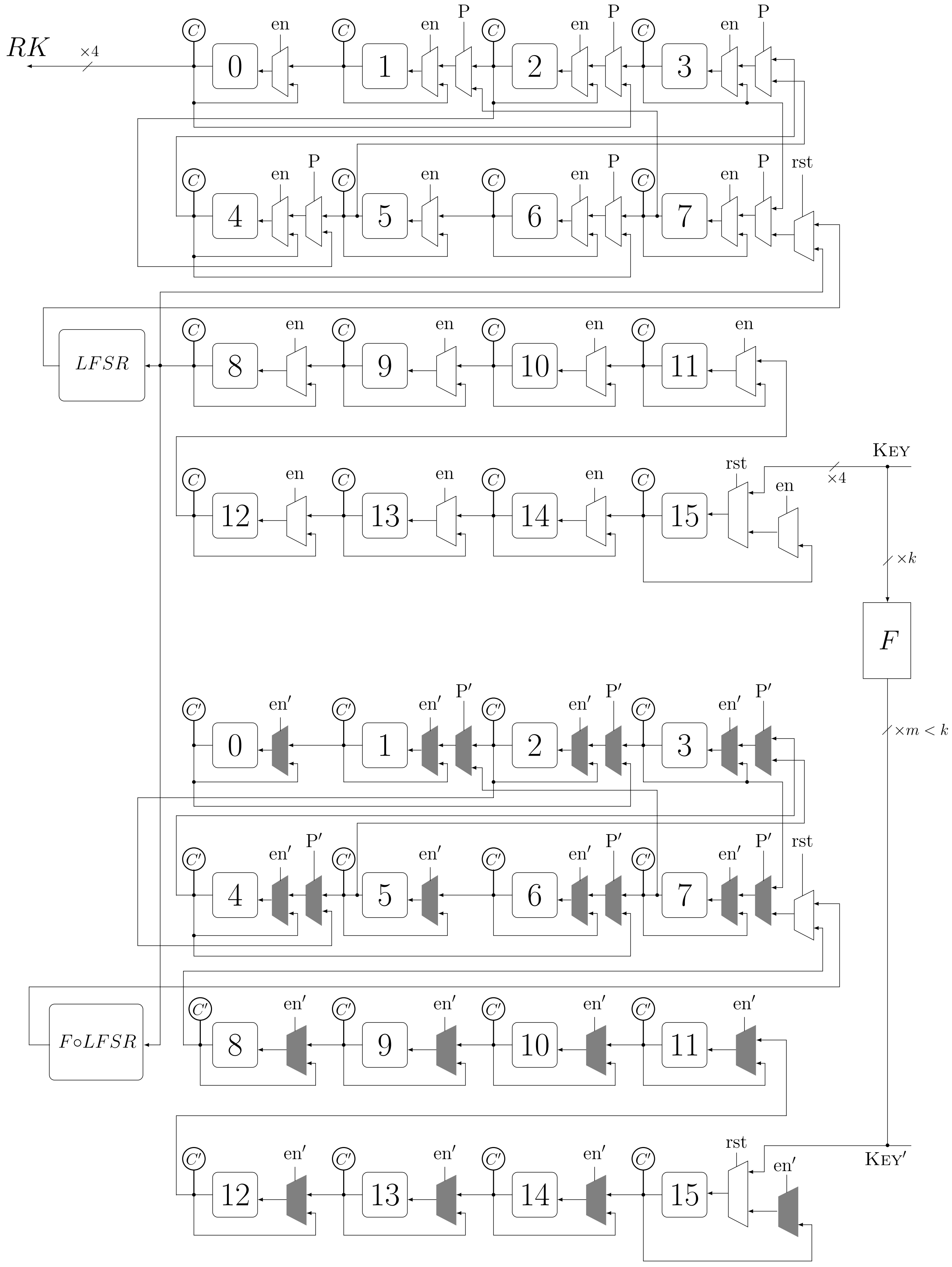
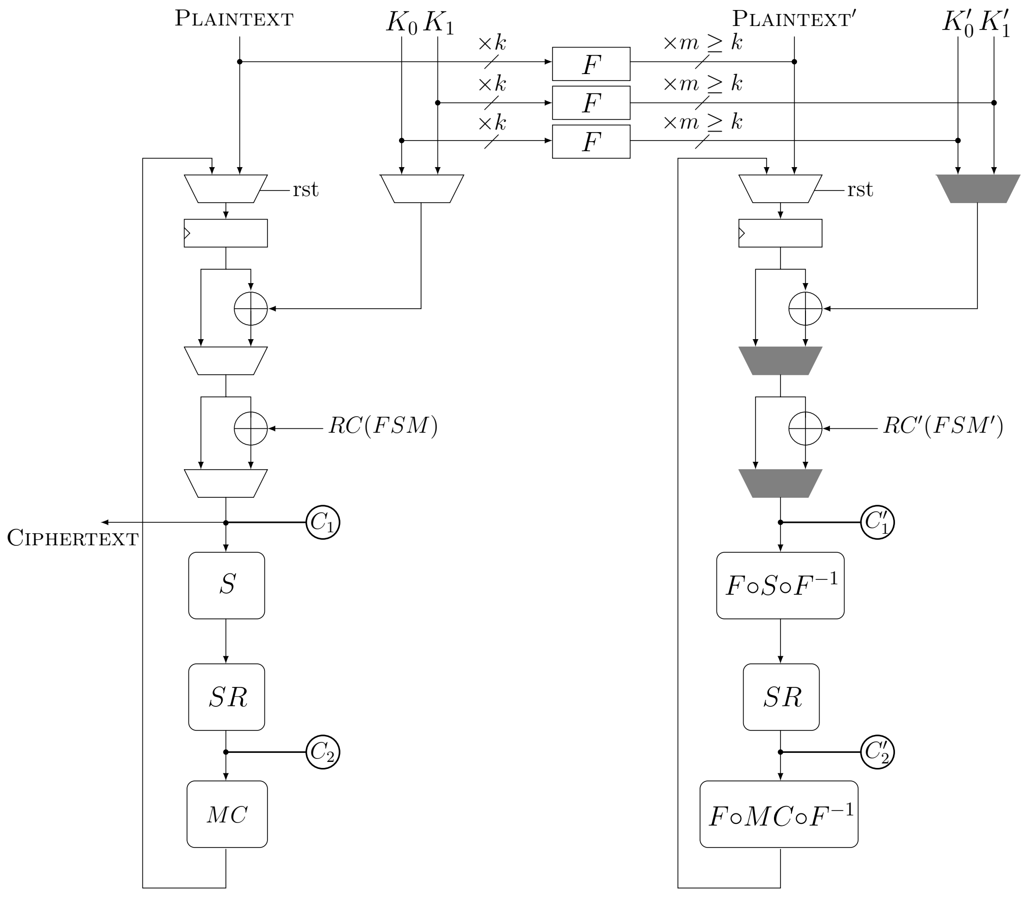
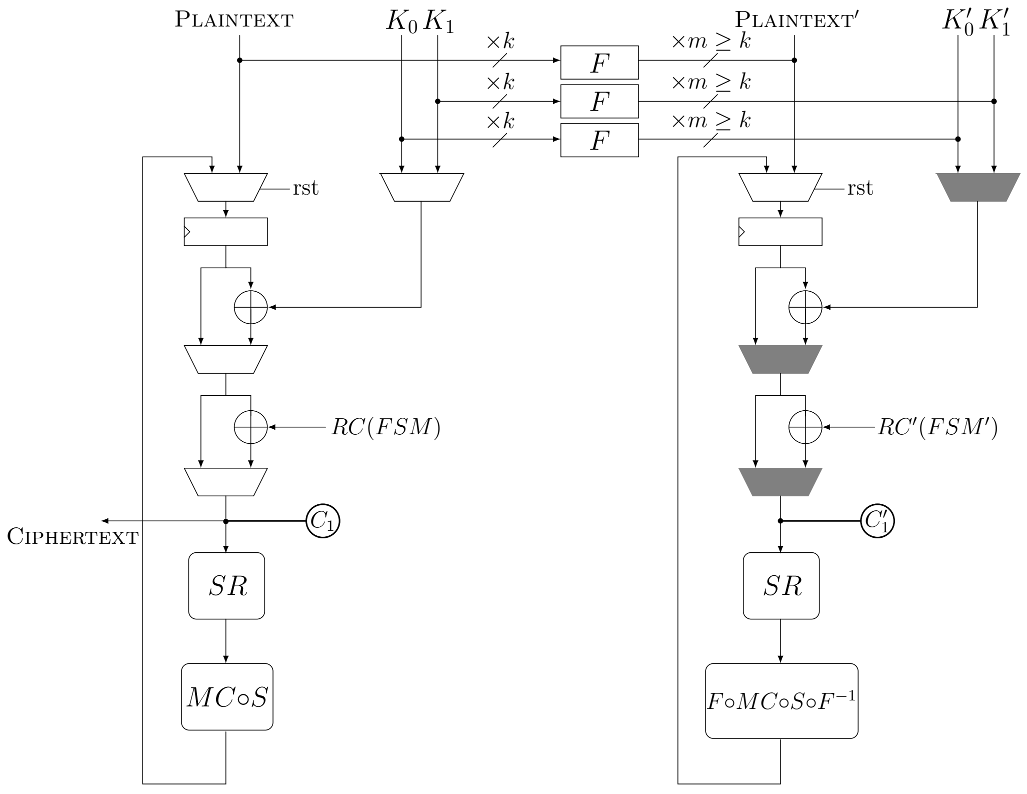


Fig. 19. Skinny, nibble-serial, key path, $m < k$.

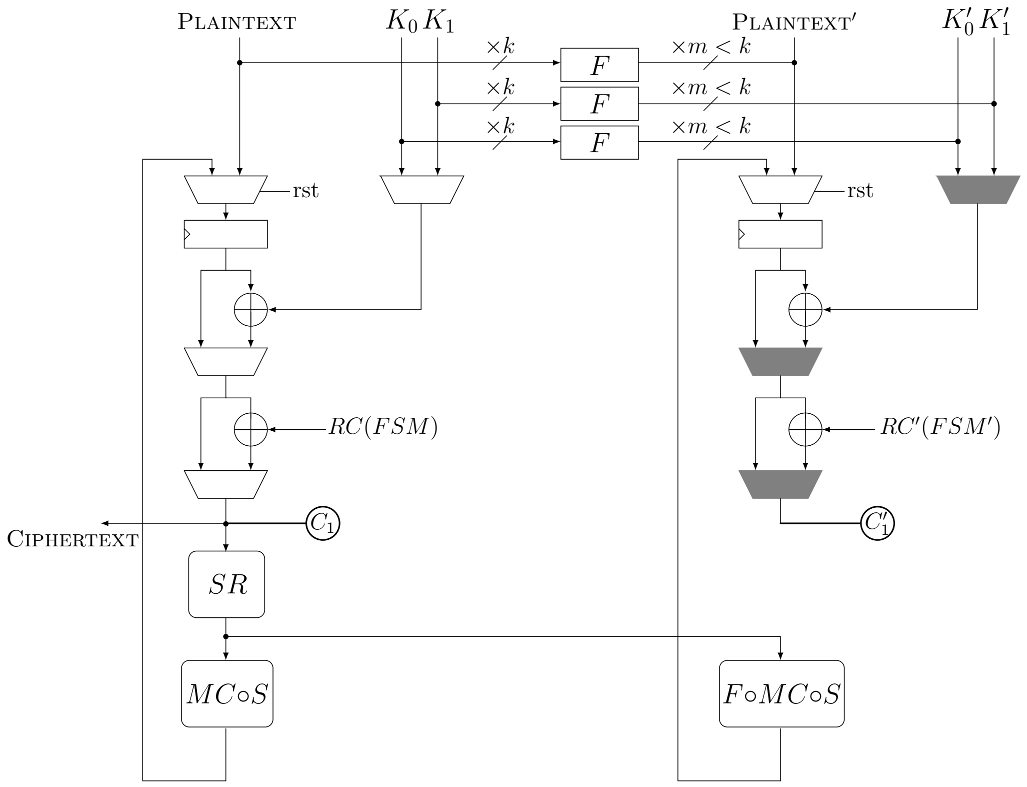


(a) 2check

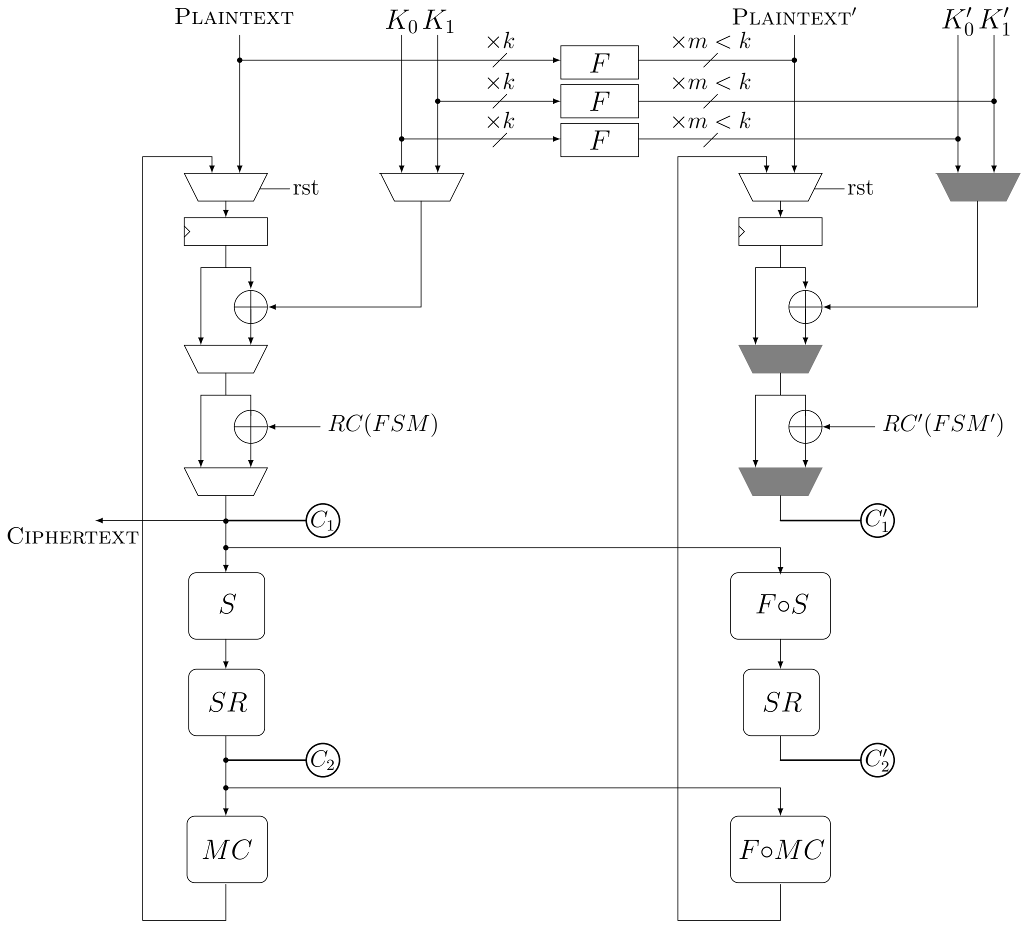


(b) combine

Fig. 20. LED, round-based, $m \geq k$.



(a) 2check



(b) combine

Fig. 21. LED, round-based, $m < k$.

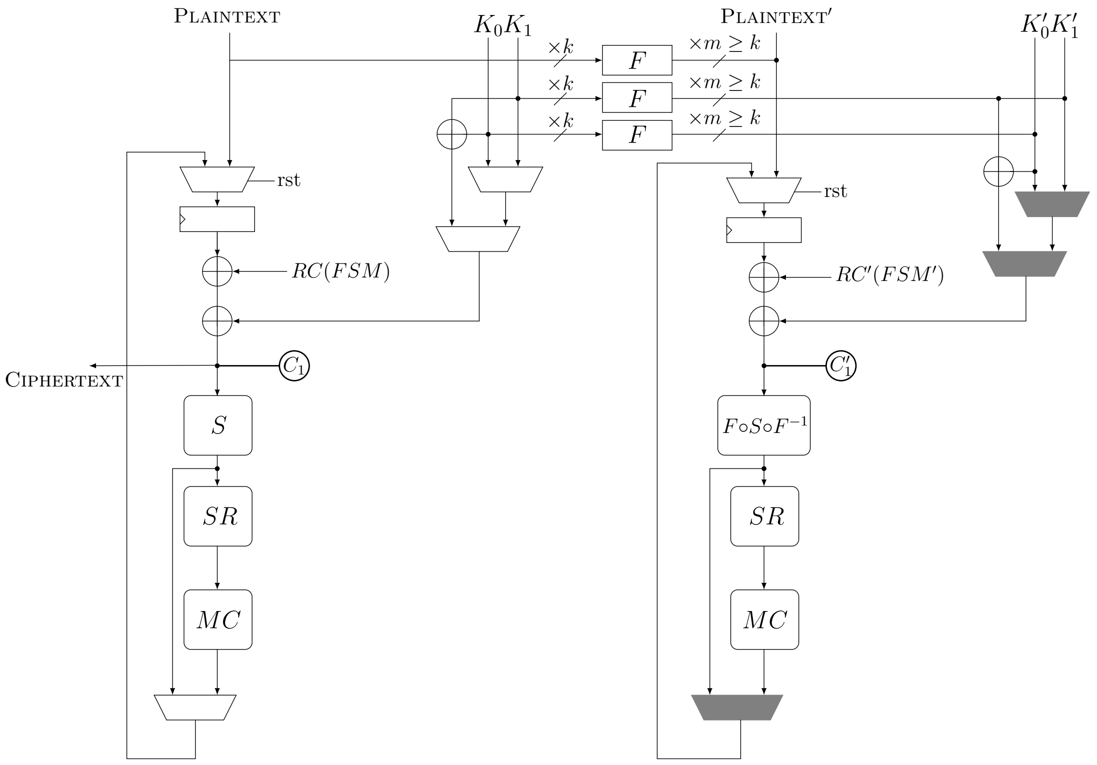


Fig. 22. Midori, round-based, $m \geq k$.

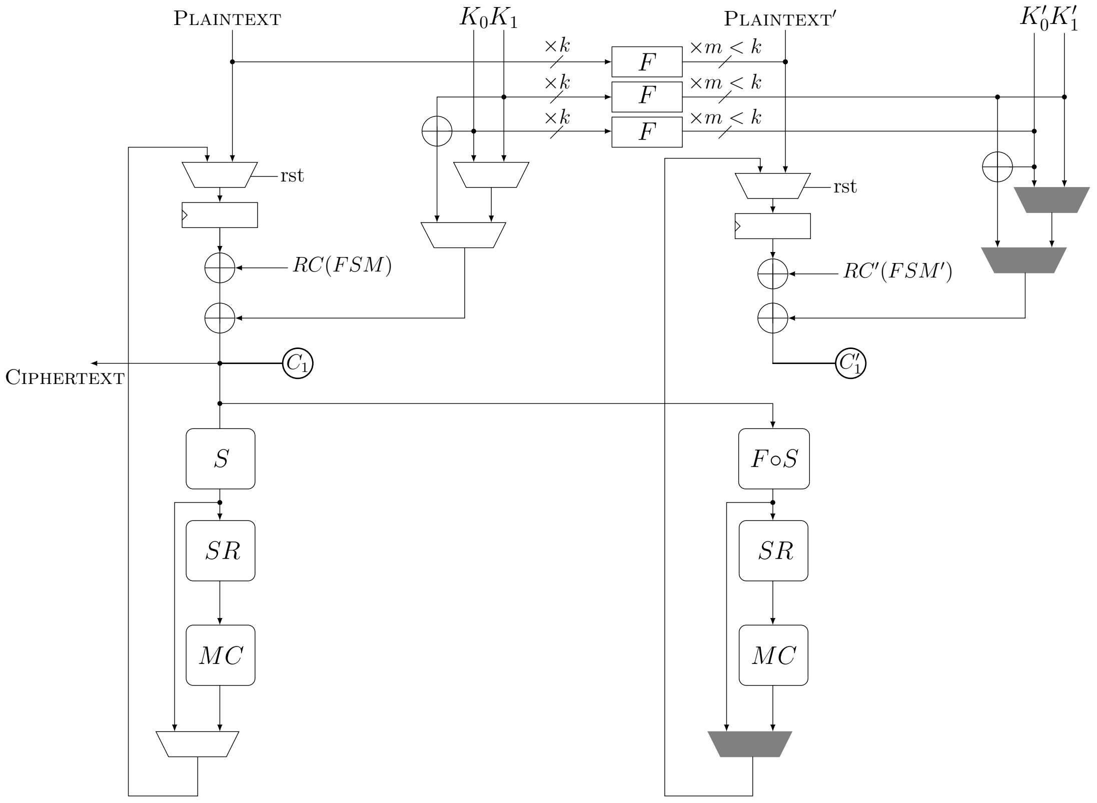


Fig. 23. Midori, round-based, $m < k$.

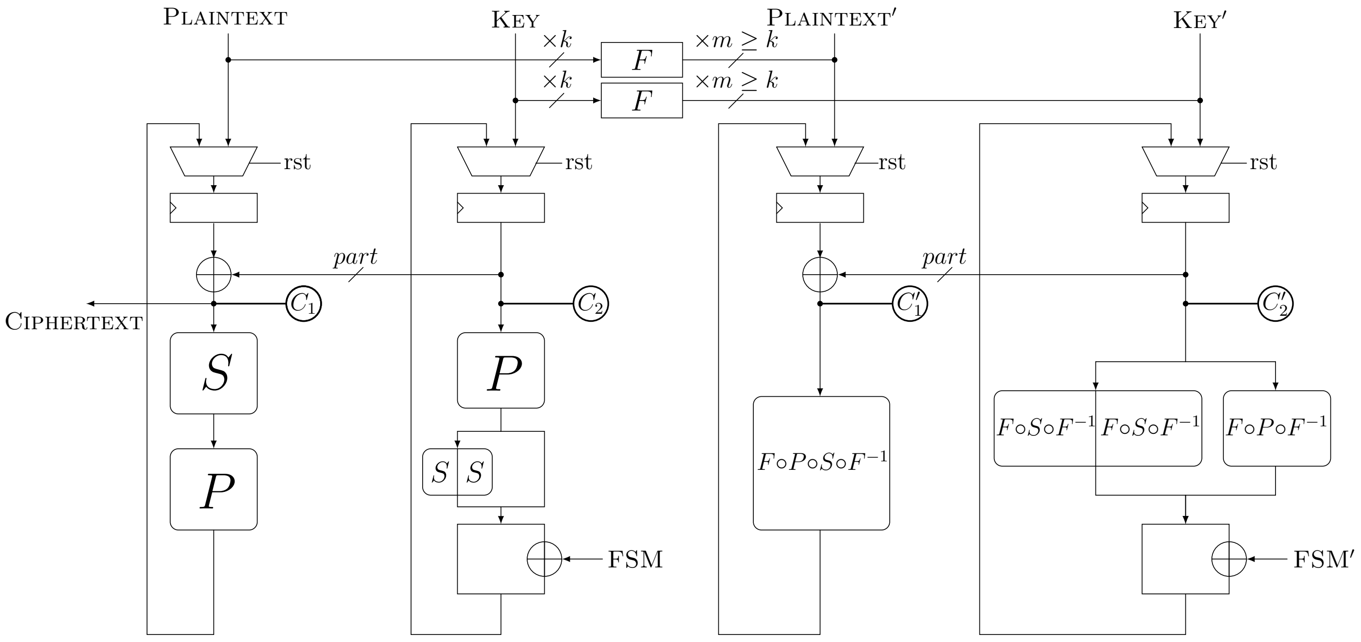


Fig. 24. PRESENT, round-based, $m \geq k$.

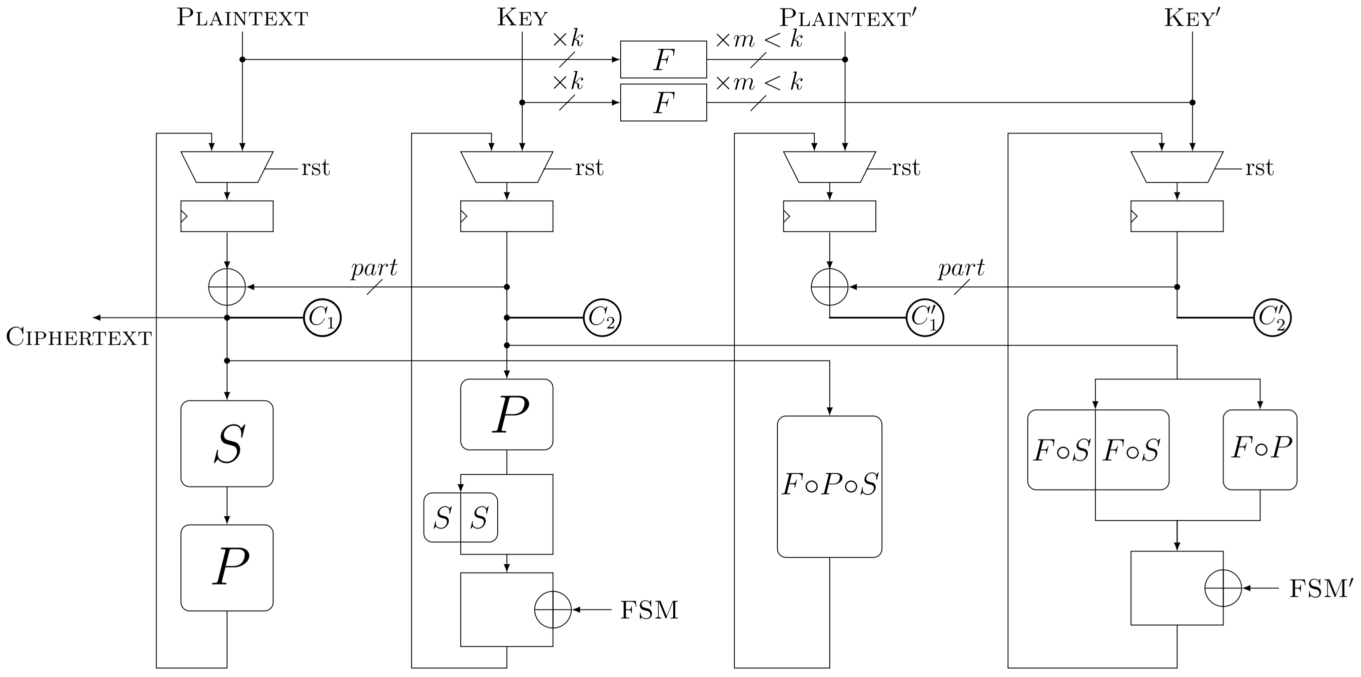


Fig. 25. PRESENT, round-based, $m < k$.

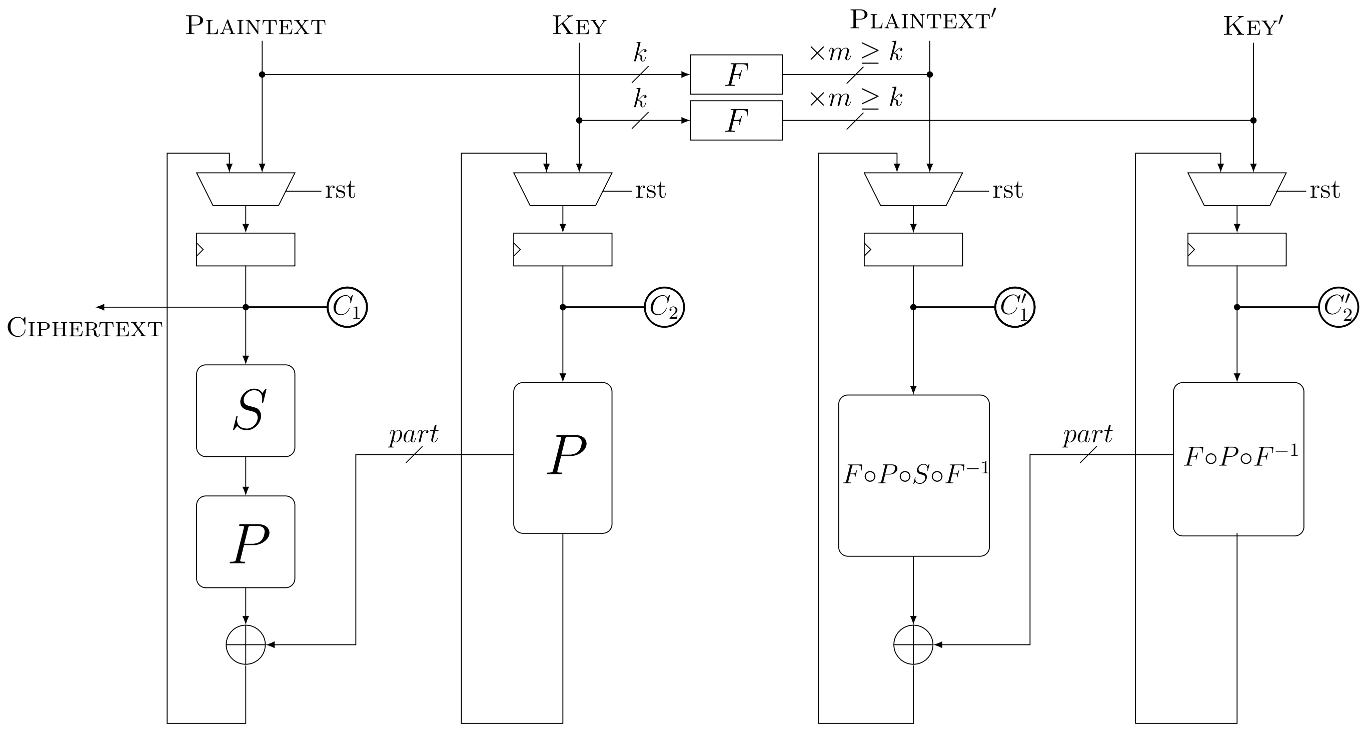


Fig. 26. GIFT, round-based, $m \geq k$.

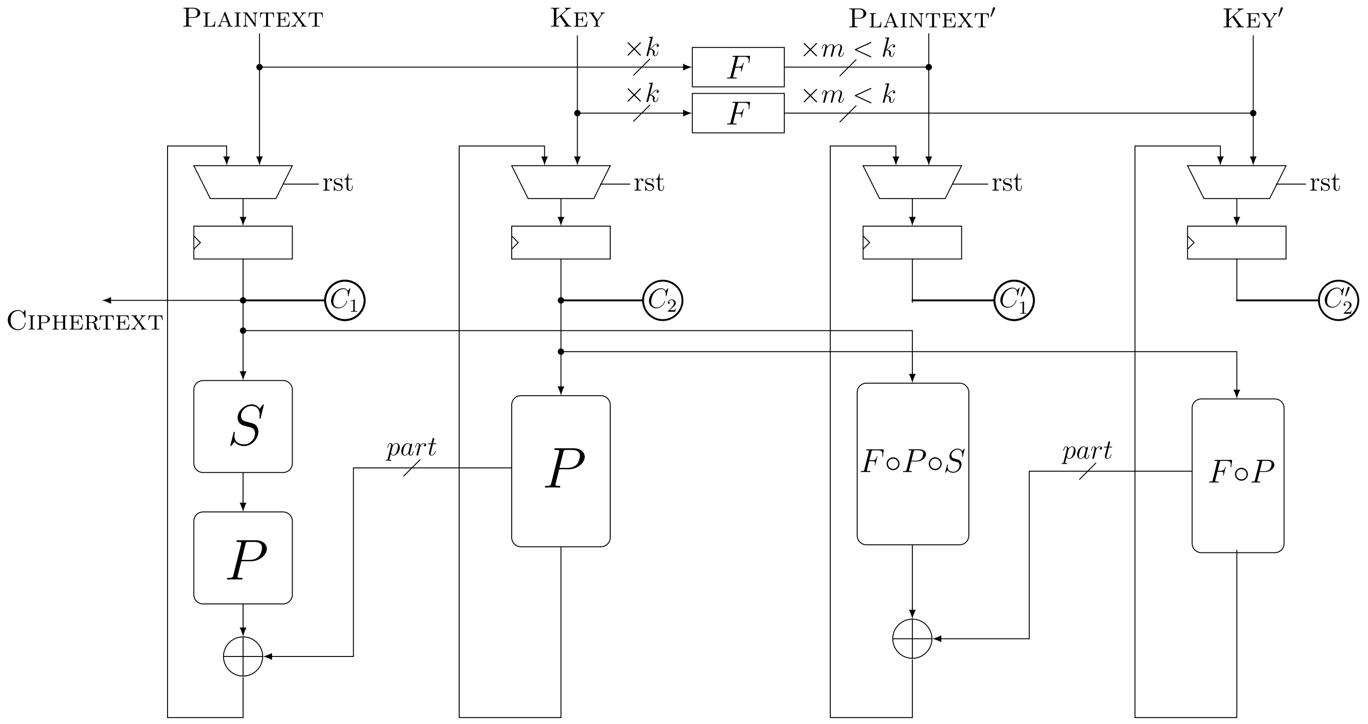


Fig. 27. GIFT, round-based, $m < k$.

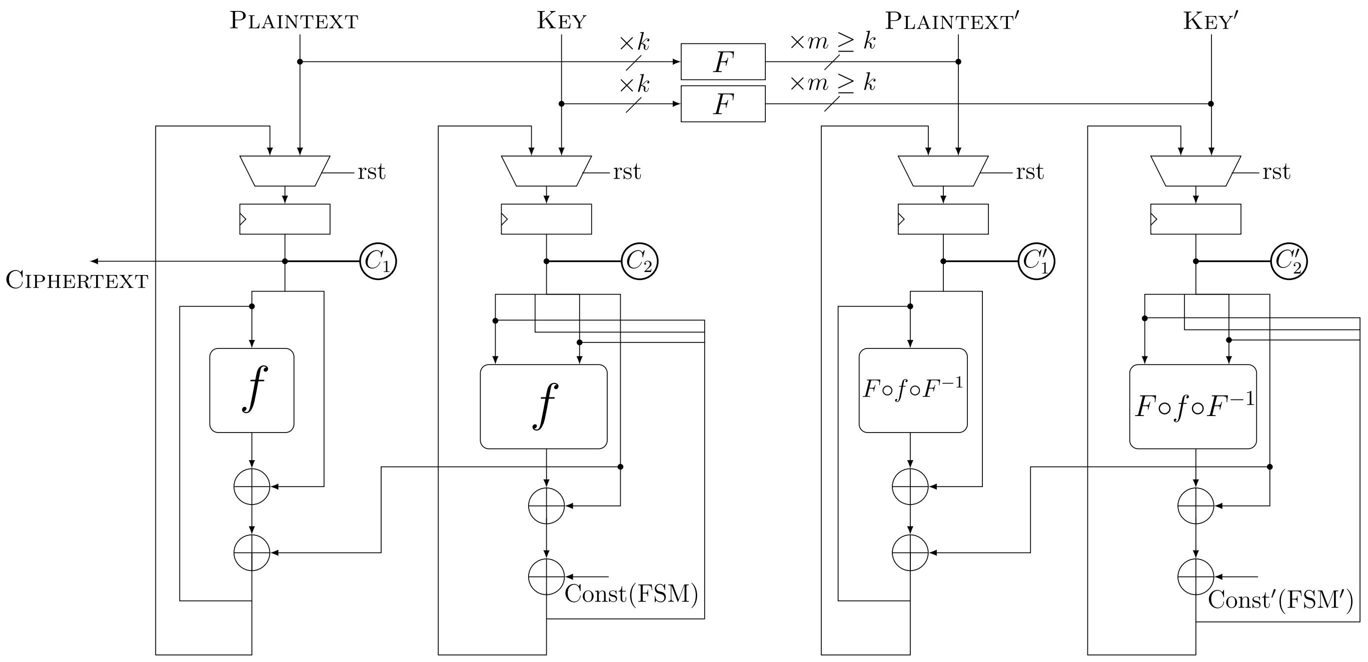


Fig. 28. SIMON, round-based, $m \geq k$.

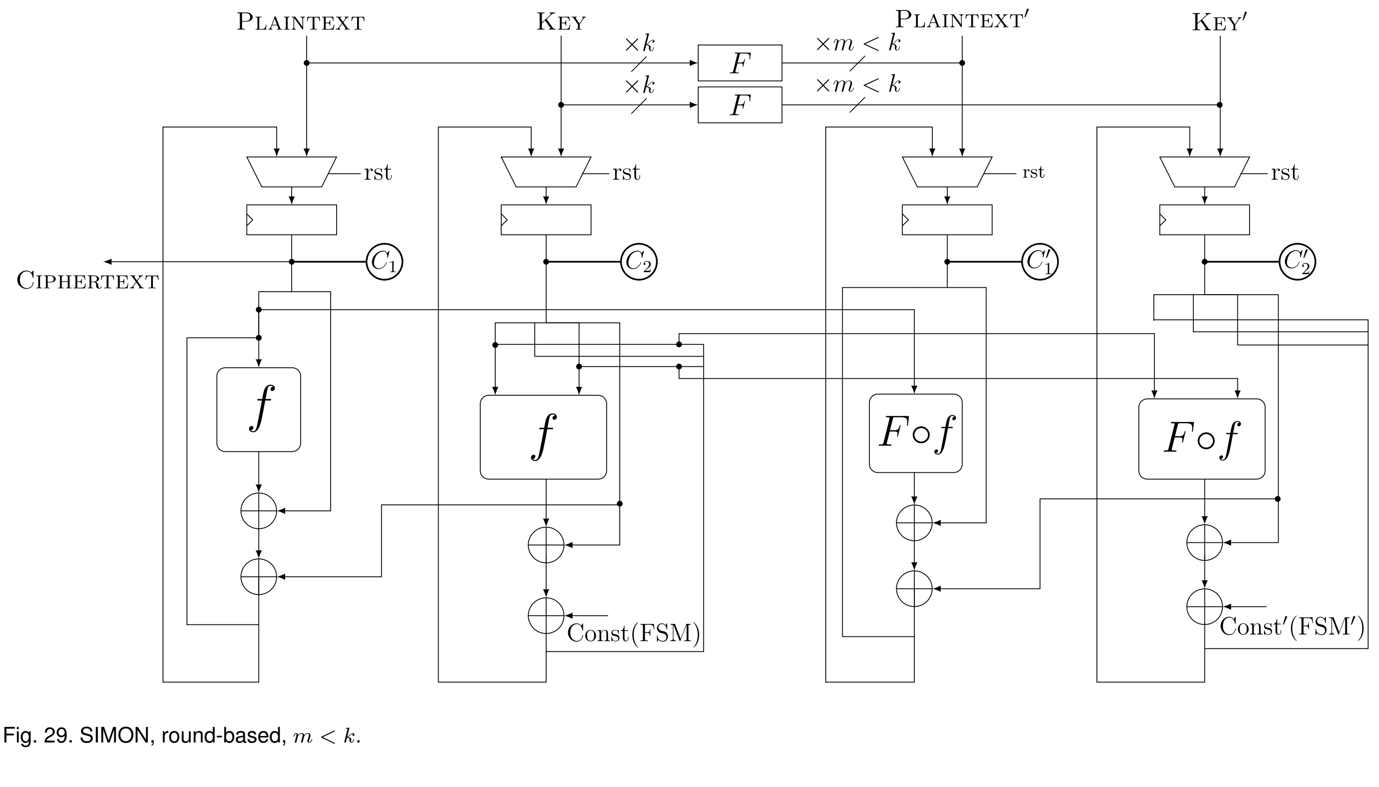


Fig. 29. SIMON, round-based, $m < k$.