Impeccable Circuits III

Shahram Rasoolzadeh
Radboud University
Digital Security Group
Nijmegen, The Netherlands
shahram.rasoolzadeh@ru.nl

Aein Rezaei Shahmirzadi
Ruhr University Bochum
Horst Görtz Institute for IT Security
Bochum, Germany
aein.rezaeishahmirzadi@rub.de

Amir Moradi
Ruhr University Bochum
Horst Görtz Institute for IT Security
Bochum, Germany
amir.moradi@rub.de

***Abstract*—As a recent fault-injection attack, SIFA defeats most of the known countermeasures. Although error-correcting codes have been shown effective against SIFA, they mainly require a large redundancy to correct a few bits. In this work, we propose a hybrid construction with the ability to detect and correct injected faults at the same time. We provide a general implementation methodology which guarantees the correction of up to t_c-bit faults and the detection of at most t_d faulty bits. Exhaustive evaluation of our constructions, by the open-source fault diagnostic tool `VerFI`, indicate the success of our designs in achieving the desired goals.**

I. Introduction

Steadily increasing the number of small embedded devices involved in our daily life attracted a considerable amount of attention towards their security. Physical accessibility to these devices enables the attacker to mount all sorts of physical threats including fault-injection attacks. The attacker forces the target device to operate in a non-regular condition and tries to recover the key by analyzing the faulty and/or fault-free outcomes. The target device can be disturbed by means of clock glitch, voltage glitch, and electromagnetic pulses while more precise faults can be injected using laser beams.

There is a considerable body of works in literature focusing on fault-injection attacks, firstly introduced by Boneh et al. [6] on RSA. By presenting a new technique called Differential Fault Analysis (DFA), Biham and Shamir successfully recovered the secret key of a DES implementation, using both faulty and fault-free ciphertexts. Fuhr et al. [12] improved this technique later by statistical analysis of only faulty ciphertexts, known as Statistical Fault Attack (SFA), leading to more relaxed requirements.

While the aforementioned techniques require faulty outputs to mount a successful attack, Ineffective Fault Attack (IFA) [8] keeps a distance to this necessity. The only requirement of IFA is the knowledge that the injected fault was effective or not. In IFA, the fault should be injected precisely and the exact location of the fault should be known, which makes it hard to mount in practice. By combining the principles of IFA and SFA, the authors of [11] proposed Statistical Ineffective Fault Attack (SIFA), which uses only fault-free ciphertexts in the key recovery process and an exact location for fault injections is not needed. These minimal requirements enable the adversary to bypass most of the known countermeasures [10], [11].

Countermeasures to fault attacks have been discussed by a great number of articles, all of which employ some sort of redundancy. The consistency of the outcome can be checked by calculating the output twice, using area/time redundancy [13]. Parity code has been used in several publications (e.g., [19]) to protect a design. Considering software platforms, more sophisticated linear codes have been employed (e.g., in [3]) while the authors of [1] took hardware implementations into account. Based on the code-based Concurrent Error Detection (CED) schemes, they proposed a methodology that guarantees the detection of any bounded faults injected in any location of the design. Notably, none of the aforementioned countermeasures defeats SIFA [10].

Naturally, a couple of techniques have been proposed to provide protection against SIFA, some focusing on error detecting, others on error correcting. Breier et al. [7] suggested an approach based on binary repetition code and majority voting. Saha et al. [17] combined the majority voting with masking and proposed a two-phase countermeasure called Transform-and-Encode. Daemen et al. [9] demonstrated that the majority voting circuitry can be replaced with a simple detection module if the entire circuit is masked using reversible building blocks, e.g., Toffoli gates. Baksi et al. [4] presented a duplication-based countermeasure, in which the state bits are randomized to remove the statistical bias caused by ineffective faults. Shahmirzadi et al. [18] extended the idea of [1] to employ Error Correcting Codes (ECCs) in Concurrent Error Correction (CEC) to prevent SIFA.

On the other hand, DFA is still a great threat to cryptographic implementations when the number of faulty encryptions is limited. Examples include the works presented in [14]–[16] where the adversary can recover the full secret key by performing only two faulty encryptions. Generally, protection against SIFA would provide resistance against DFA as well, but fault models in SIFA are usually limited to a single (or a few) bits while DFAs generally cover larger fault models. At the dark side, extending the countermeasures of SIFA to cover DFA would potentially lead to a high overhead. For example, 3-bit redundancy is required to correct a single-bit fault in a 4-bit value, i.e., protecting against 1-bit SIFA. In

The work described in this paper has been supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972 and the project 406956718 SuCCESS. Besides, S. Rasoolzadeh is supported by the Netherlands Organisation for Scientific Research (NWO) under TOP grant TOP1.18.002 SCALAR.

order to protect against a DFA with 2-bit faults, this leads to 7 redundant bits when extending the correction to 2 bits. As a matter of fact, no study offers a hybrid construction with the ability to detect and correct injected faults at the same time. To complete this missing piece of the puzzle, this paper provides a clear guideline of how such a construction should be implemented in hardware platforms.

Our Contributions: In this paper, we propose a new methodology based on code-based schemes providing secure hardware implementation in which fault propagation is considered. First of all, we demonstrate that following the injective structure proposed in [1], [18] is neither necessary nor beneficial, and unite them into a single structure improving the area overhead. Then, we consolidate the principle of error-detection method in [1] with the error-correcting facility of [18] by keeping the same adversary model. Our constructions guarantee the correction of up to t_c erroneous bits and the detection of t_d-bit faults given that $t_d > t_c$. We protect every component of the circuit including data path, FSM, and cipher's key schedule at any clock cycle. We applied our methodology to `CRAFT` [5] to have a fair comparison to the state of the art in terms of area overhead and latency. We verified the ability of our constructions in correcting and detecting the faults using the open-source fault-diagnostics tool VerFI [2]. In short, it confirms our claims as long as the injected faults fit into the considered bounded model.

II. Preliminaries

Error Detecting Codes (EDCs) and ECCs are essential aspects of information theory and are often used in the countermeasures against fault-injection attacks. In the following, the necessary notions related to these codes are recalled.

Definition 1 (Binary Code)**.** *A binary $[n, k]$-code C with $n > k$, is a bijective mapping from the space of messages $\mathcal{M} = \mathbb{F}_2^k$ to the space of codewords $\mathcal{C} \subset \mathbb{F}_2^n$, i.e., each message $x \in \mathcal{M}$ is mapped to a unique codeword $c \in \mathcal{C}$ with $c = C(x)$.*

The parameters n and k are referred to as the length *and* rank *of the $[n, k]$-code C, respectively. Besides,* parity size *refers to the difference between length and rank, i.e., $n-k$.*

Definition 2 (Systematic Code)**.** *A code in which the message x is embedded in the codeword c is called a systematic code, i.e., the codeword c is the concatenation of x with a parity (redundancy) x', i.e., $c = (x\|x')$, while the parity bits are generated from x.*

Systematic codes enable a simple split of the data paths between message and parity. Therefore, the original implementation of the target operation can stay as it is. Furthermore, the decoder can take the first k bits of a codeword to extract the message, i.e., no implementation cost.

Definition 3 (Linear Code)**.** *The $[n, k]$-code for which the codeword space is a vector subspace over $\mathbb{F}_2^n$ is called* linear.

Focusing on systematic linear codes does not lead to any restrictions, since any linear non-systematic code can be transformed into a systematic code with the same properties.

Definition 4 (Generator Parity Check Matrices and Syndrome)**.** *For a linear $[n, k]$-code, the $k \times n$ matrix G that maps a message to the corresponding codeword, is called the* generator matrix*, i.e., $C(x) = x \cdot G$.*

Since the rank of the generator matrix is k, there are $n-k$ linear equations between the codeword bits to be satisfied. These equations can be shown as matrix multiplication. The $n \times (n-k)$ matrix H that checks if an element of $\mathbb{F}_2^n$ is a possible codeword is called the parity check matrix. *Besides, for any $x \in \mathbb{F}_2^n$, the output of $x \cdot H$ is called* syndrome. *Hence, for any $c \in \mathcal{C}$, the corresponding syndrome $c \cdot H = 0^{n-k}$.*[1]

The generator matrix G of a linear systematic $[n, k]$-code is of the form $G = [I_k|P]$ with I_k the identity matrix of size k, while the parity bits are generated using a $k \times (n-k)$ matrix P as $x' = x \cdot P$. Then, for any $(x\|x') \in \mathbb{F}_2^n$, the syndrome can be computed by $x \cdot P \oplus x'$.

Definition 5 (Minimum Distance)**.** *The minimum distance d of an $[n, k]$-code C is defined as*

$$d = \min_{\forall c, c' \in \mathcal{C},\ c \neq c'} \mathrm{hw}(c \oplus c'),$$

where hw *denotes the Hamming weight. An $[n, k]$-code with minimum distance d is denoted as an $[n, k, d]$-code.*

Lemma 1. *For a linear $[n, k]$-code, $d = \min_{\forall c \in\ \mathcal{C}\backslash\{0\}} \mathrm{hw}(c)$.*

Lemma 2. *A code C with minimal distance d can detect additive errors $e \in \mathbb{F}_2^n$, if $\mathrm{hw}(e) < d$, i.e., the faulty codeword $\tilde{c} = c \oplus e$. Moreover, such a code can correct the errors e if $\mathrm{hw}(e) < d/2$.*

Proof. Clearly, each $\tilde{c} \in \mathbb{F}_2^n$ which $\tilde{c} \notin \mathcal{C}$ is erroneous. In the error detection mode, for any $c \in \mathcal{C}$ and any $e \in \mathbb{F}_2^n$ with $\mathrm{hw}(e) < d$, we have $c \oplus e \notin \mathcal{C}$. Hence, such an error is detectable.

Besides, in the error correction mode, for each $\tilde{c} \in \mathbb{F}_2^n$, there is only one $c \in \mathcal{C}$ with $\mathrm{hw}(\tilde{c} \oplus c) < d/2$. Otherwise, it contradicts the definition of minimum distance. Therefore, for any $c \in \mathcal{C}$ and any $e \in \mathbb{F}_2^n$ with $\mathrm{hw}(e) < d/2$, the erroneous word $\tilde{c} = c \oplus e$ can be corrected to the codeword c. □

Syndrome Decoding is an efficient method of error detection or error correction in linear codes. The underlying principle idea lies on the linearity of the code. Naturally, if the value of the syndrome is zero, the given input is a possible codeword; otherwise, not. Assuming a systematic linear code, for a codeword $(x\|x') \in \mathcal{C}$, we have $x' \oplus x \cdot P = 0$. Then, for an erroneous codeword $(x \oplus e\|x' \oplus e')$, the syndrome is $e'' = (x' \oplus e') \oplus (x \oplus e) \cdot P = e' \oplus e \cdot P$. Using a proper look-up table, one can map all syndrome values to the error vector $(\tilde{e}\|\tilde{e}')$ with $\tilde{e}' \oplus \tilde{e} \cdot P = e''$. Such a look-up table $e'' \mapsto (\tilde{e}\|\tilde{e}')$ is called a *syndrome decoder*. Considering that in the coding channels, occurrence probability of errors with smaller Hamming weight

[1] 0^ℓ and 1^ℓ denote the ℓ-bit array of all 0 and all 1, respectively.

is higher, for a given syndrome value, the syndrome's error is chosen from the ones that have the minimum possible Hamming weight, i.e., $\mathrm{hw}(\tilde{e}) + \mathrm{hw}(\tilde{e}') < d/2$.

By adding the syndrome's error $(\tilde{e}\|\tilde{e}')$ to the erroneous codeword, it is possible to correct the codeword. In this way, output of addition is $(x \oplus e \oplus \tilde{e} \| x' \oplus e' \oplus \tilde{e}')$ and if the occurred error $(e\|e')$ is the same as the syndrome's error $(\tilde{e}|\tilde{e}')$ (which is the case if $\mathrm{hw}(e) + \mathrm{hw}(e') < d/2$), the output is the correct codeword $(x\|x')$.

It is important to mention that the statement in Lemma 2 does not mean both correcting the errors with $\mathrm{hw}(e) < d/2$ and detecting the other faults with $\mathrm{hw}(e) < d$ is possible. It can only detect errors with $\mathrm{hw}(e) < d$ or only correct the errors with $\mathrm{hw}(e) < d/2$. The following lemma, generalizes the correcting and detecting capability of the codes.

Lemma 3. *A code C with minimal distance d can correct all errors $e \in \mathbb{F}_2^n$ with $\mathrm{hw}(e) \leq t_c$ and detect all errors e with $t_c < \mathrm{hw}(e) \leq t_d$, if $t_c + t_d < d$.*

Proof. Since $t_c < t_d$ and $t_c + t_d < d$, hence $t_c < d/2$. Therefore, for any $c \in \mathcal{C}$ and any $e \in \mathbb{F}_2^n$ with $\mathrm{hw}(e) \leq t_c$, the closest codeword to the erroneous word $\tilde{c} = c \oplus e$ is c and it is unique. Therefore, $\tilde{c} = c \oplus e$ with $\mathrm{hw}(e) \leq t_c$ is uniquely corrected to the codeword c. We call the subset $\{c \oplus e | c \in \mathcal{C}, \mathrm{hw}(e) \leq t_c\}$ *the correction subset.*

For any c and any e with $t_c < \mathrm{hw}(e) \leq t_d$, the minimum distance of the erroneous word $\tilde{c} = c \oplus e$ from all codewords is larger than t_c, since $t_c + t_d < d$. Therefore, such an erroneous word is out of the correction subset, but can be detected. □

Here after, for simplicity, we use the *t_c-bit correction and t_d-bit detection* notion to call the correction up-to t_c bits and detection the other faults up-to t_d bits. Therefore, for a t_c-bit correction and t_d-bit detection, we need to use a code with minimum distance of at least $t_c + t_d + 1$.

Example 1. *The Extended Hamming code $[8, 4, 4]$ is capable of either 1) detecting up-to 3-bit errors or 2) correcting all 1-bit errors and detecting all 2-bit errors. Hence, we can use it in a $t_c = 1$-bit correction and $t_d = 2$-bit detection scheme.*

III. Methodology

A. Adversary Model

We assume the same adversary models as in [18], where the adversary can inject faults at t arbitrarily cells (either a register or a gate) by flipping or setting the cell output to a certain value.

Definition 6 (Univariate Adversary Model $\mathcal{M}_t$)**.** *In a given sub-circuit, the adversary can make at most t cells faulty in the entire operation of the algorithm, e.g., a full encryption. t can be split into various clock cycles.*

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

B. Fault Propagation and Independence Property

If an input of a gate in a circuit is faulty, depending on the type of the gate and the value of the other inputs to the gate, its output might become faulty. This phenomenon is propagated through the circuit and as a result an $\mathcal{M}_t$-bounded adversary can achieve t' faulty cells, where $t' \geq t$. The affect of fault propagation is well-studied in [1]. It is shown that an $\mathcal{M}_t$-bounded attacker can target t certain cells in such a way that more than t faults appear at the sub-circuit output avoiding the underlying code to detect or correct it.

To prevent fault propagation, *independence property* has been defined in [1]. It means that the corresponding sub-circuits for computing each output bit must be implemented separately, where no cell is shared between sub-circuits. Hence, any injected fault in one cell, can only affect at most one output bit.

C. Correction and Detection Point

In order to correct up-to t_c-bit faults and detect the other faults up-to t_d bits, we use a similar approach to the *correction point* introduced in [18]. The construction shown in Fig. 1 demonstrates the correction+detection general structure.

We use $F : \mathbb{F}_2^k \mapsto \mathbb{F}_2^m$ with $F(x) = x \cdot P$ to refer to the multiplication with matrix P, where $m = n - k$ denotes the parity (redundancy) bit-length. As explained in Section II, we make use of an $[n, k, t_c + t_d + 1]$-code, and write $\forall x \in \mathbb{F}_2^k, x' \in \mathbb{F}_2^m$ with $\mathrm{hw}(x) + \mathrm{hw}(x') \leq t_c$ and $\tilde{x} = F(x) \oplus x'$, we have

$$SD_1(\tilde{x}) = x, \;\; SD_2(\tilde{x}) = x', \;\text{ and }\; g(\tilde{x}) = 0,$$

while for any other syndrome value $\tilde{x}$, we have

$$SD_1(\tilde{x}) = 0, \;\; SD_2(\tilde{x}) = 0, \;\text{ and }\; g(\tilde{x}) = 1.$$

Assume a faulty input codeword $(x \oplus e \| x' \oplus e')$ with the injected fault $(e\|e')$. At the correction+detection point, the syndrome decoder is fed by $F(x \oplus e) \oplus x' \oplus e' = F(e) \oplus e'$; hence, the injected fault $(\tilde{e}\|\tilde{e}')$ is predicted. If $\mathrm{hw}(e) + \mathrm{hw}(e') \leq t_c$, the predicted fault is the same as the injected one; hence adding $(\tilde{e}|\tilde{e}')$ to the input word would eliminate the injected fault. Note that in this case, the fault detector function $g(.)$ stays 0. If $t_c < \mathrm{hw}(e) + \mathrm{hw}(e') \leq t_d$, the output of the syndrome decoders would be 0 while $g(.)$ becomes 1. It is important to recall that the syndrome values for $\mathrm{hw}(e) + \mathrm{hw}(e') \leq t_c$ are always different to those for $t_c < \mathrm{hw}(e) + \mathrm{hw}(e') \leq t_d$, i.e., no collision between correcting and detecting.

It is noteworthy to mention that the syndrome decoders SD_1 and SD_2 are employed to correct faulty bits in original faults x and its redundancy x', respectively. Since we are only interested in correcting faulty bit(s) in x, we only implement SD_1 to achieve lower area overhead while maintaining the same security level (see Fig. 2(b)). More details will be given in the next subsection.

D. Application

Any sequential circuit can be represented by the general scheme depicted in Fig. 2(a), which we use to to apply our correction+detection strategy. It consists of a register which loads the Input at the beginning (triggered by rst signal)

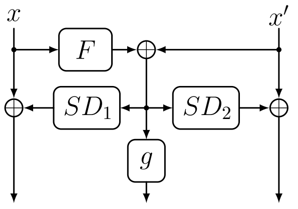
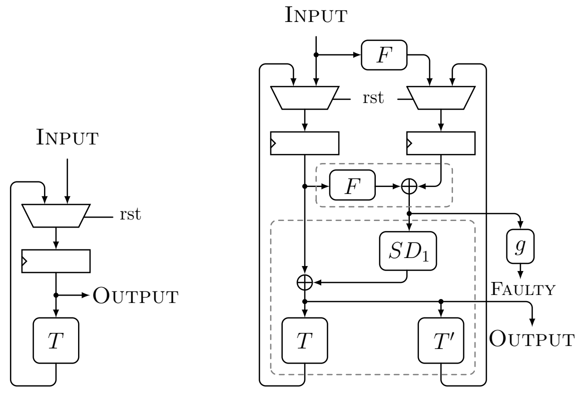


Fig. 1. Correction+detection point.



(a) Original Structure (b) Protected Structure

Fig. 2. Our proposed construction for both fault correction and detection.

and performs the function T repeatedly until the OUTPUT is taken from the register. First, we only consider a univariate adversary model, and later extend it to cover the multivariate case. In contrary to the Concurrent Error Detection (CED) and Concurrent Error Correction (CEC) schemes in [1] and [18], which depending on the injective-ness of the F function different structures were introduced, here, we only propose one general structure for any F. Later, we explain that this general structure covers all cases of [1] and [18].

Our correction+detection structure is shown in Fig. 2(b). The redundant function T' is achieved by $T' = F \circ T$ and it is independent of the injective-ness of F. Therefore, since T' needs to receive the original data x, we only need to correct faults on x. That means, we do not need to implement the SD_2 function, i.e., less area overhead.

Note that, it is necessary to place a correction+detection point at the input of each operation. Otherwise, the faults injected at the register cells would potentially propagate to multiple output bits of T or T'. Besides, all output bits of each dashed boxes in Fig. 2(b) must be implemented fulfilling the independence property which may necessitate implementing several instances of SD_1.

Further, in a code with $d = t_c + t_d + 1$, the output of SD_1 does not change if up to t_c faults are injected. Thus, F and the corresponding XOR can be instantiated separately which can be beneficial to limit the area overhead. In short, our structure guarantees CEC security against $\mathcal{M}_{t_c}$ and CED security against $\mathcal{M}_{t_d}$ adversary models.

E. Extension to Multivariate Adversary

To extend our construction to a univariate model, we follow the concept suggested in [18]. More precisely, we need to use a code with a distance of $2(t_c + t_d) + 1$. This is because between two consecutive correction+detection points, a $\mathcal{M}_t^*$ adversary is able to inject $2t$ faults. Therefore, by applying an $[n, k, 2(t_c + t_d) + 1]$-code, we guarantee CEC security against $\mathcal{M}_{t_c}^*$ and CED security against $\mathcal{M}_{t_d}^*$ adversary models.

F. FSM, Control Signals, Multiplexers, and Output

a) Multiplexers: Assume a k-bit multiplexer switching between x and y controlled by a signal s. To equip it with t_c-bit correction and t_d-bit detection, in theory, it is enough to make a t_c-bit correcting multiplexer by following the methodology described in [18], and check the consistency of the signal s, which is able to detect up to t_d-bit faults. Any uncorrected fault in the multiplexer will be detected at the next detection point. In practice, however, this might lead to an insecure design due to the limited drive strength of a cell, as every cell has a certain maximum fanout. For a k-bit multiplexer, k instances of 2-to-1 multiplexer should be used, all of which are controlled by the same signal s. Therefore, buffers are placed at the output of the cell driving the signal s. Hence, some 2-to-1 multiplexers are controlled by the original signal s and others by the buffered signal. The attacker can take advantage of this and inject fault into the buffers and bypass the consistency check as the original signal s is fault-free. Therefore, we need to design a protected multiplexer which is able to correct t_d-bit faults to ensure that the attacker cannot gain any advantage by injecting fault to any buffers.

To this end, we employ a repetition code to protect the selector signal s. Namely, the control signal s is generated $2t_d + 1$ times following the independence property; their concatenation can be seen as $(s \| s')$, where s' is the redundancy with the length of $2t_d$ bits. Considering x' and y' as the parity (redundant counterpart) of x and y, the error correcting multiplexer is a k+m-bit multiplexer switching between $(x \| x')$ and $(y \| y')$ and controlled by $(s \| s')$. It is indeed a multiplexer tree with $2t_d + 1$ levels, where the first row is controlled by s, and the other rows by s'. The inputs v_i of the first multiplexer row with $0 \leq i < 2^{k+m}$ are defined as follow:

$$v_i = \begin{cases} (x \| x') & ; \quad i = 0^{2t_d+1} \oplus \delta \text{ , hw}(\delta) \leq t_d \\ (y \| y') & ; \quad i = 1^{2t_d+1} \oplus \delta \text{ , hw}(\delta) \leq t_d \\ \quad 0 & ; \quad else \end{cases}$$

b) Output: We assume that the circuit provides a control signal DONE demonstrating the end of e.g., encryption process. An attacker can target such a signal and obtain intermediate values of the cipher. Due to the issue described above for the control signal s, checking the consistency of the DONE signal with its redundant counterpart DONE$'$ is not enough in practice. Hence, to avoid this kind of sniffing, we can make use of a construction similar to the multiplexer. More precisely, the DONE is generated t_d+1 times, and concatenated with FAULTY signal (see Fig. 2(b)) to form a t_d+2-bit signal controlling a t_d+2-level multiplexer. Note that FAULTY $= 1$ indicates that no fault is detected, i.e., invert of the function $g(.)$ in Fig. 1. The first-row inputs of the multiplexer are defined as:

$$v_i = \begin{cases} \textsc{Output} & ; \quad i = 1^{t_d+2} \\ 0 & ; \quad else \end{cases}$$

This construction guarantees to protect the design against sniffing intermediate results by up to t_d faults. Notably the implementation should fulfill the independence property.

c) FSM and Control Signals: An attacker can manipulate the circuit's flow by inducing faults on the FSM, hence obtaining intermediate results of the cipher exploiting the secrets. The implementation of every FSM contains a set of registers called STATE that is initialized by INIT and updated every clock cycle by an update function U. The construction shown in Fig 2(b) can be applied to such a circuit as well by replacing INPUT and T with INIT and U, receptively. As a part of the control logic, suppose that a function G_i receives the corrected STATE and provides a control signal s_i, e.g., for a multiplexer. Since every part of the circuit controlled by FSM can be constructed by a multiplexer, we utilize the above-given repetition code (i.e., $2t_d + 1$) on every control signal. Namely, for each i, the function G_i is instantiated multiple times following the independence property.

IV. Case Study

Optimized with respect to protection against DFA, CRAFT [5] is a lightweight block cipher operating on a 64-bit state, 128-bit key, and 64-bit tweak within 32 rounds. Based on the given key and tweak, the key schedule generates 4 tweakeys, one of which is chosen every round based on the round counter. The round function is made out of five involutory round operations: SB, MC, PN, AC, and ATK. While the state is shown by a matrix of 4×4 nibbles, in each round, the MC matrix is multiplied to each column of nibbles; then the AC and the ATK are applied to the state followed by the SB and the PN operations. Note that SB and PN are missing in the last round. To assess the area overhead and latency of our proposed methodology, we focus on round-based implementation of encryption-only module, i.e., without tweak and decryption. We synthesized our designs by Synopsys Design Compiler and all simulations and reports are based on NanGate 45 nm ASIC standard cell library.

A. Implementation Details

Since the CRAFT's S-box is applied on nibbles, we considered a code with $k = 4$ and $d = 5$ to correct 2-bit faults, and another code with the same message size but $d = 4$ to correct 1-bit fault and detect 2-bit faults. More precisely, we applied $[11, 4, 5]$-code for the former and $[8, 4, 4]$-code for the latter case. We can actually use larger codes, i.e., with larger message sizes k but with the same distance d. To point out such a trade-off, we also applied $[22, 16, 4]$-code to equip CRAFT with 1-bit correction and 2-bit detection, while encoding 16-bit messages. The results and comparison of our implementations are summarized in Table I. We should highlight that the number of clock cycles to perform an encryption in the all designs is 32 and the critical path of each design is reported as latency in that table.

TABLE I
Area and Latency Comparison of CRAFT Round-based Implementations, using NanGate 45 nm ASIC Library.

	Code	Area (GE)	Latency (ns)	Ref.
unprotected	$[4, 4, 1]$	1097	0.55	[5]
1-bit corr.	$[7, 4, 3]$	5187	0.87	[18]
2-bit corr.	$[11, 4, 5]$	21617	1.08	[18]
	$[11, 4, 5]$	17538	1.10	Sec. IV
1-bit corr.	$[8, 4, 4]$	7799	0.97	Sec. IV
+2-bit det.	$[22, 16, 4]$	7438	1.28	Sec. IV

$[11, 4, 5]$-*code:* Here, the F function is injective and the parity generator matrix P is

$$P_{[11,4,5]} = \begin{bmatrix} 1\,1\,1\,1\,0\,0\,0 \\ 1\,1\,0\,0\,1\,1\,0 \\ 1\,0\,1\,0\,1\,0\,1 \\ 0\,1\,0\,1\,0\,1\,1 \end{bmatrix}.$$

We followed the general structure shown in Fig. 2(b), even though F is injective. Compared to [18], utilizing the non-injective construction leads to around 20% less area overhead and roughly the same latency while maintains the same security claim, i.e., 2-bit correction (see Table I).

$[8, 4, 4]$-*code:* This code – known as *Extended Hamming* code – is capable of correcting 1-bit fault and detecting 2-bit faults with the following parity generator matrix.

$$P_{[8,4,4]} = \begin{bmatrix} 0\,1\,1\,1 \\ 1\,0\,1\,1 \\ 1\,1\,0\,1 \\ 1\,1\,1\,0 \end{bmatrix}$$

Due to the issue mentioned earlier in Section III-F, we implemented a protected multiplexer which is able to correct 2-bit faults on its controlling (select) signal to ensure that the attacker cannot gain any advantage by injecting fault to any buffers. Since $k = 4$ (i.e., dealing with nibbles), the flow of the construction is similar to the previous code. As shown in Table I, compared to $[7, 4, 3]$-code, which can only correct 1-bit faults, the area overhead of increasing its capability to detect 2-bit faults is around 50%.

$[22, 16, 4]$-*code:* The correction and detection capability of this code is the same as $[8, 4, 4]$-code, as both have the same distance $d = 4$. Its parity generator matrix can be written as

$$P_{[22,16,4]} = \begin{bmatrix} 0\,0\,0\,0 & 0\,1\,0\,1 & 0\,1\,1\,1 & 0\,1\,1\,1 \\ 0\,0\,0\,0 & 1\,0\,1\,0 & 1\,0\,1\,1 & 1\,0\,1\,1 \\ 0\,1\,1\,1 & 1\,1\,1\,1 & 0\,0\,0\,0 & 1\,1\,0\,1 \\ 1\,0\,1\,1 & 0\,0\,0\,0 & 0\,0\,0\,0 & 1\,1\,1\,0 \\ 1\,1\,0\,1 & 0\,0\,1\,1 & 1\,1\,0\,1 & 0\,0\,0\,0 \\ 1\,1\,1\,0 & 1\,1\,0\,0 & 1\,1\,1\,0 & 0\,0\,0\,0 \end{bmatrix}.$$

Here, each 16-bit cipher state has 6 parity bits, i.e., 10 bits less compared to $[8, 4, 4]$-code. However, F and SD_1 are larger functions. The implementation of the MC, SB operations, XORs, and the FSM is similar to the case in $[8, 4, 4]$-code. However, PN cannot solely operate on the redundant counterpart as the state is not coded nibble-wise. As a result, PN should be applied before T and T' operations (see Fig. 2(b)). As shown in Table I, compared to $[8, 4, 4]$-code, this trick can slightly reduce the area overhead (around 5%), but it has a negative impact on the circuits latency (32% larger).

B. *On the Choice of* $[22, 16, 4]$*-code*

The above-given $P_{[22,16,4]}$ matrix has been chosen in order to reduce the area of the implementation. We searched through all 6×16 binary matrices with $d=4$ with the following criteria:

- Hamming weight of P should be minimum, which in this case is 48. This condition is reflected by the implementation of the F function, since an implementation of an $r \times c$ binary matrix M following the independence property needs $\mathrm{hw}(M)-r$ number of XORs.
- Considering each row of P as 4 nibbles, the number of nonzero nibbles should be minimum. This is related to the implementation of $F \circ S$. As an example, let us consider the implementation of its last output bit, which is related to the last row of P, i.e., `1110,1100,1110,0000`. It means that `1110` is multiplied to the output of the first Sbox, and so on. Therefore, the fourth Sbox does not contribute to the last output bit of $F \circ S$. Looking at P, each row has at most 3 nonzero nibbles.

C. *Implementation of* F

In contrast to other part of the design, we can implement the F function, which encodes the given INPUT (see Fig. 2(b)), without considering the independence property. This improvement is due to the cryptanalytic security of the cipher. Consider $(x\|x')$ and $(y\|y')$ as the codeword corresponding to the plaintext x and ciphertext y. Assume that the adversary injects some faults (bounded by the underlying code) in the aforementioned F function (without independence property). This results in $(x\oplus e_i\|x'\oplus e'_i)$ with probable fault propagation, i.e., $\mathrm{hw}(e_i)+\mathrm{hw}(e'_i)$ is not necessarily smaller than d. There are three possible cases for $(x \oplus e_i\|x' \oplus e'_i)$:

1) It is a valid codeword; it will pass through the first correction+detection point without any changes.
2) It is not a codeword, but it is in the correction zone; it will be changed to $(\hat{x}\|\hat{x}')$ which may equal $(x\|x')$.
3) It is not a codeword and also not in the correction zone; the first correction+detection point will detect the fault.

In the first two cases, the fault is neither corrected nor detected. Since any other (t_c, t_d)-bound fault will be corrected or detected, we just need to make sure that the adversary cannot have a successful attack in those two cases. Here, the output of the first correction+detection point is a valid codeword $(\hat{x}\|\hat{x}')$ (different than $(x\|x')$) leading to $(\hat{y}\|\hat{y}')$ as a a valid codeword for the generated ciphertext. Any successful attack with the knowledge of $\big((x\|x'),(y\|y')\big)$ and $\big((\hat{x}\|\hat{x}'),(\hat{y}\|\hat{y}')\big)$ pairs (even with many of such two-pairs) is equal to a successful differential cryptanalysis attack with the knowledge of having (several of) x, y and $\hat{x}$, $\hat{y}$ plaintext/ciphertext pairs. Therefore, the adversary gains no advantage by injecting faults into F functions, if the cipher is secure against differential cryptanalysis. This improvement is also valid for the CED and CEC schemes of [1] and [18].

D. *Simulation*

To verify our proposed constructions, we employed the bit-sliced implementation of the open-source fault-diagnostics tool `VerFI`(ver2Beta) [2]. For a fixed plaintext and key, the tool injects either bit-flip or stuck-at-0/1 into specified gates at desired clock cycle(s). It receives the NanGate 45 net-list of the design and provides a report presenting the number of detected, non-detected, and ineffective faults.

We examined our protected `CRAFT` designs equipped with $[11, 4, 5]$-code considering two different adversary models: i) injecting all possible 2-bit faults at a certain clock cycle. i.e., $\mathcal{M}_{t=2}$, and ii) injecting all possible 2-bit faults split into two consecutive clock cycles, i.e., $\mathcal{M}^*_{t=1}$. The design contains 11 394 cells and the tool running on a 40-core machine with two Intel Xeon CPUs required 14 and 26 minutes respectively to report the correction of all injected faults. The designs equipped with $[8, 4, 4]$-code and $[22, 16, 4]$-code consist of 4 331 and 4 148 cells, respectively. These designs have been evaluated in two scenarios: i) all possible single-bit faults at a clock cycle, i.e., $\mathcal{M}_{t=1}$, and ii) all possible 2-bit faults at a clock cycle, i.e., $\mathcal{M}_{t=2}$. Conducting each of these fault simulations took less then 2 minutes, reporting the correction of all 1-bit and detection of all 2-bit faults.

V. Conclusions

In this work, we introduced a new structure for both fault correction and detection in the cryptographic primitives. By applying the code-based schemes, we showed how to correct up to t_c-bit injected faults and at the same time detect other faults up to t_d bits (with $t_d > t_c$). Further, we presented several improvements compared to the state-of-the-art schemes. Our construction guarantees the correction and detection of (t_c,t_d)-bounded faults injected in any point of the circuit including the data path, FSM, control signals, and even the modules responsible for faults correction/detection and at any clock cycle. As a case study, we applied our technique on `CRAFT` block cipher and compared the performance figures with the known only-correction designs. Based on the evaluations conducted using the fault diagnostic tool `VerFI`, we claim protection against SIFA with up to t_c faults and against DFA with up to t_d-bit injected faults.

References

[1] A. Aghaie, A. Moradi, S. Rasoolzadeh, A. R. Shahmirzadi, F. Schellenberg, and T. Schneider, "Impeccable Circuits," *IEEE Trans. Computers*, vol. 69, no. 3, pp. 361–376, 2020.

[2] V. Arribas, F. Wegener, A. Moradi, and S. Nikova, "Cryptographic Fault Diagnosis using VerFI," in *HOST 2020*. IEEE, 2020.

[3] 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.

[4] A. Baksi, V. B. Y. Kumar, B. Karmakar, S. Bhasin, D. Saha, and A. Chattopadhyay, "A novel duplication based countermeasure to statistical ineffective fault analysis," in *ACISP 2020*, ser. LNCS, vol. 12248. Springer, 2020, pp. 525–542.

[5] C. Beierle, G. Leander, A. Moradi, and S. Rasoolzadeh, "CRAFT: Lightweight Tweakable Block Cipher with Efficient Protection Against DFA Attacks," *IACR ToSC*, vol. 2019, no. 1, pp. 5–45, 2019.

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

[7] J. Breier, M. Khairallah, X. Hou, and Y. Liu, “A Countermeasure Against Statistical Ineffective Fault Analysis,” *IACR Cryptology ePrint Archive*, vol. 2019, p. 515, 2019.

[8] C. Clavier, “Secret External Encodings Do Not Prevent Transient Fault Analysis,” in *CHES*, ser. LNCS, vol. 4727, 2007, pp. 181–194.

[9] J. Daemen, C. Dobraunig, M. Eichlseder, H. Groß, F. Mendel, and R. Primas, “Protecting against statistical ineffective fault attacks,” *IACR TCHES*, no. 3, pp. 508–543, 2020.

[10] 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*, ser. LNCS, vol. 11273, 2018, pp. 315–342.

[11] C. Dobraunig, M. Eichlseder, T. Korak, S. Mangard, F. Mendel, and R. Primas, “SIFA: Exploiting Ineffective Fault Inductions on Symmetric Cryptography,” *IACR TCHES*, vol. 2018, no. 3, pp. 547–572, 2018.

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

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

[14] D. Mukhopadhyay, “An Improved Fault Based Attack of the Advanced Encryption Standard,” in *AFRICACRYPT*, ser. LNCS, vol. 5580, 2009, pp. 421–434.

[15] G. Piret and J. Quisquater, “A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD,” in *CHES*, ser. LNCS, vol. 2779, 2003, pp. 77–88.

[16] D. Saha, D. Mukhopadhyay, and D. R. Chowdhury, “A Diagonal Fault Attack on the Advanced Encryption Standard,” *IACR Cryptol. ePrint Arch.*, p. 581, 2009.

[17] S. Saha, D. Jap, D. B. Roy, A. Chakraborty, S. Bhasin, and D. Mukhopadhyay, “A Framework to Counter Statistical Ineffective Fault Analysis of Block Ciphers Using Domain Transformation and Error Correction,” *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 1905–1919, 2020.

[18] A. R. Shahmirzadi, S. Rasoolzadeh, and A. Moradi, “Impeccable Circuits II,” in *DAC 2020*. IEEE, 2020, pp. 1–6.

[19] K. Wu, R. Karri, G. Kuznetsov, and M. Gössel, “Low Cost Concurrent Error Detection for the Advanced Encryption Standard,” in *ITC*. IEEE Computer Society, 2004, pp. 1242–1248.