Dynamic encoding, a lightweight combined countermeasure against hardware attacks

Maxime Montoya, Simone Bacles-Min, Anca Molnos, Jacques J.A. Fournier

Dynamic encoding, a lightweight combined countermeasure against hardware attacks

Maxime Montoya*, Simone Bacles-Min†, Anca Molnos‡, Jacques J.A. Fournier‡
* NXP Semiconductors Austria GmbH
Gratkorn, Austria
E-mail: firstname.lastname@nxp.com
† Univ. Grenoble Alpes, CEA, LIST
MINATEC Campus, F-38054 Grenoble, France
E-mail: firstname.lastname@cea.fr
‡ Univ. Grenoble Alpes, CEA, LETI
MINATEC Campus, F-38054 Grenoble, France
E-mail: firstname.lastname@cea.fr

***Abstract*—With the Internet of Things (IoT) an increasing amount of sensitive data have to be communicated and hence encrypted. Low-cost hardware attacks such as fault analysis (FA) or side-channel analysis (SCA) threaten the implementation of cryptographic algorithms. Many countermeasures have been proposed against either of these attacks, however, only a few countermeasures protect efficiently an implementation against both attacks. These combined countermeasures usually have a prohibitive area and power overhead, and require up to thousands of bits of fresh randomness at each encryption. Therefore, they may not be suited to protect lightweight algorithms in resource-constrained devices. In this paper, we propose a new combined countermeasure, which is particularly adapted to protect lightweight algorithms based on shift registers. It achieves an efficient power balancing at algorithmic level, and provides an inherent fault detection with a better coverage than most existing combined countermeasures. Furthermore, it has a smaller power and area overhead than existing combined countermeasures, and requires at most 8 random bits at each encryption.**

I. Introduction

The number of connected devices has grown exponentially over the past few years and 20 billion such devices are expected by 2020 [1]. These devices process and communicate an ever increasing amount of sensitive data such as credentials, software updates, or personal information, which are usually encrypted to preserve their confidentiality. Many devices in the Internet of Things (IoT) are deployed in easily accessible locations, and their operation is not supervised. Therefore, the integrated circuits in these devices are a choice target for hardware attacks, that aim to recover secret keys used to encrypt data. In particular, non-destructive attacks such as fault analysis (FA) [2]–[4] or side-channel analysis (SCA) [5]–[8] can be performed on-site with low-cost material. For example, faults could be induced by injecting either voltage or clock glitches in the attacked device, and SCA usually only requires the monitoring of the external voltage of a chip. Therefore, protections against both FA and SCA have to be implemented into the integrated circuits for the IoT. Many such protections exist, which usually introduce high area, time and power overheads. However, IoT devices are typically strongly resource-constrained. For example, low-cost RFID tags usually have a small area, and battery-powered sensors require a small energy consumption. Therefore, there is a need for lightweight hardware countermeasures, both in terms of area and energy consumption.

Implementing a protection against both SCA and FA is not straightforward, as some countermeasures against one attack may facilitate or have limited resilience against the other type of attack. For example, redundancy is usually used to detect fault injections, but it increases the side-channel leakage and thus facilitates SCA [9]. Another example is dual-rail with precharge logic styles, such as WDDL [10], which inherently provide some fault detection thanks to the differential encoding of every bit. However, they offer a limited fault coverage, and a fault injected on both bits of a differential pair would be undetected. Furthermore, these logic styles usually have a high area and power overhead, and their throughput is reduced due to the use of a precharge signal.

Recently, several combined countermeasures based on the principles of *masking* have been proposed. In *ParTI* [11], masking is used together with error detection codes; in the implementation of *Private Circuits II* described in [12], it is combined with a differential encoding of every bit. Finally, in *CAPA* [13] and *Masks and Macs* [14], information theoretic MAC tags are masked together with the processed data. The solutions proposed in [12]–[14] consist in splitting a variable x into d shares x_i, such that $x = \sum_i x_i$ where $i \in \{1, \ldots, d\}$. These approaches only cover faults injected in at most $(d-1)$ shares, and are vulnerable to faults affecting simultaneously all shares, which can be induced with low-cost methods such as clock or voltage glitches [2]. In addition, the authors of [13] and [14] explain that error detection codes such as those used in *ParTI* can only detect faults up to a given Hamming weight, and that these codes are vulnerable against smart faults injections where an attacker knows which codes are used.

The implementation of these masking-based combined countermeasures is very expensive in terms of area, power consumption and throughput, and requires up to thousands of bits of fresh randomness for each encryption with the protected algorithm. This cost may be prohibitive for resource-constrained devices used in the IoT. Among these protections, the most lightweight is *Masks and Macs*, where the fault detection has by itself an area overhead factor of 2.53 compared to a first-order masking of the AES that is described in [15]. When considering also the cost of masking, *Masks and Macs* has an area overhead factor higher than 6.

In this article, we focus on the protection of lightweight security primitives based on shift registers. The stream cipher Trivium [16] is representative of these security primitives. It was among the three finalists of the ECRYPT Stream Cipher Project in 2008 [17], which aimed at selecting the most efficient and compact stream ciphers. Other security primitives such as the block cipher KATAN [18] and the hash function QUARK [19] are built upon the design of Trivium. These security primitives have an internal state of hundreds of bits, which is contained in one or several shift registers and updated at every clock cycle with simple combinational logic. In these algorithms, the power consumption and the area of combinational logic is negligible compared to that of flip flops. As a consequence, both FA [3], [4] and SCA [5]–[8] against these algorithms target the internal state contained within the shift registers and disregard the combinational logic. We propose a new combined countermeasure against both FA and SCA, which is much lighter than existing combined countermeasures and is particularly suited to protect lightweight security primitives based on shift registers. It consists of an algorithm-level power balancing without precharge that inherently detects faults, with a better fault coverage than most existing protections.

Our contributions. We present a novel combined countermeasure called *dynamic encoding* that consists in balancing the power consumption without precharge, at algorithmic level. We describe how to apply this protection to shift registers that constitute the basic building block of several lightweight security primitives (Section II). We discuss its inherent security against fault injections, and explain why it is more robust than most existing fault detection mechanisms (Section III). We propose a methodology to integrate randomness into dynamic encoding, in order to increase its security against both attacks (Section IV). Finally, we evaluate the area and power overheads of dynamic encoding when applied to a shift register and to the stream cipher Trivium (Section V), and we assess its security against side-channel attacks with various metrics (Section VI).

II. Principles and definitions

In this section, we describe the application of dynamic encoding to a shift register with an internal state S containing k bits S_i, with $i \in \{1, \dots, k\}$. All operations are performed in GF(2). To balance the power consumption of this unprotected shift register, each bit S_i of its internal state is transformed by a function $\mathcal{F}$ to a n-bit code $\mathcal{F}(S_i)$, with $n > 1$. These codes have to respect the following:

Property 1. *Each bit S_i of the internal state is encoded with the same Hamming weight (HW):*
$\exists C \mid \forall i \in \{1, \dots, k\},\ HW(\mathcal{F}(S_i)) = C.$

Property 2. *Two consecutive bits of the internal state are encoded with a constant Hamming distance (HD):*
$\exists C \mid \forall i \in \{1, \dots, k-1\},\ HD(\mathcal{F}(S_i), \mathcal{F}(S_{i+1})) = HW(\mathcal{F}(S_i) + \mathcal{F}(S_{i+1})) = C.$

This encoding is implemented by applying $\mathcal{F}$ to the input bit x of the unprotected shift register, at each clock cycle. Similarly, the inverse function $\mathcal{F}^{-1}$ is applied to the encoded output bit $\mathcal{F}(z)$ of the protected shift register, to obtain a 1-bit output z. Therefore, for an n-bit code $\mathcal{F}(x)$, n parallel shift registers contain the encoded internal state. In the remaining of this paper, we denote with *share* each shift register in parallel and a n-bit code $\mathcal{F}(x)$ is decomposed into n shares. Note that these *shares* are not to be confused with those manipulated with the *masking* countermeasure against side-channel attacks. On the contrary to masking, dynamic encoding is not related to *secret sharing*.

A well-known encoding is the differential encoding, which is used in dual-rail logic styles. It consists, for instance, in encoding a bit "0" into "01", and a "1" into "10". Though it respects Property 1, it requires another mechanism, such as a precharge, to fulfill Property 2. We denote this encoding as a *static encoding*, as a single bit "0" (or "1") is always encoded the same way.

Definition. *Given two consecutive bits x and y, dynamic encoding consists in encoding these two bits with two different codes, for any values of x and y. These two different codes have to respect both Property 1 and Property 2.*

Corollary. *The codes generated by such a dynamic encoding must be at least 4-bit long: for any bit x, $\mathcal{F}(x) \in GF(2^n)$ with $n \geq 4$.*

Proof. Property 2 implies that two consecutive identical bits are encoded differently with $\mathcal{F}$. Therefore, there are at least two codes to represent a "0", and two other codes to represent a "1". According to Property 1, these four codes must have the same HW. The codes that respect this property are only two in $GF(2^2)$, i.e., $\{10, 01\}$, and three in $GF(2^3)$, i.e., either $\{001, 010, 100\}$ or $\{110, 101, 011\}$. Starting with $GF(2^4)$, there are at least four such codes, e.g., $\{0001, 0010, 0100, 1000\}$. The HD between any two of these four codes is 2, and hence they also respect Property 2.

For the remaining of this paper, we only consider dynamic encoding in $GF(2^4)$ with two codes to represent a "1" and two other codes to represent a "0". Dynamic encoding could also be realized in $GF(2^4)$ with the six codes $\{0011, 1100, 1001, 0110, 1010, 0101\}$, or in $GF(2^n), n > 4$. However, these solutions incur a higher overhead and we expect them to provide no additional resistance against either

TABLE I: Examples of encoding tables, for (a) a 1-of-4 dynamic encoding and (b), (c) two different 2-of-4 dynamic encodings

(a)

$\mathcal{F}(\mathbf{x},\mathbf{c})$	$\mathbf{c}=\mathbf{0}$	$\mathbf{c}=\mathbf{1}$
$\mathbf{x}=\mathbf{1}$	1000	0100
$\mathbf{x}=\mathbf{0}$	0010	0001

(b)

$\mathcal{F}(\mathbf{x},\mathbf{c})$	$\mathbf{c}=\mathbf{0}$	$\mathbf{c}=\mathbf{1}$
$\mathbf{x}=\mathbf{1}$	1001	1010
$\mathbf{x}=\mathbf{0}$	0110	0011

(c)

$\mathcal{F}(\mathbf{x},\mathbf{c})$	$\mathbf{c}=\mathbf{0}$	$\mathbf{c}=\mathbf{1}$
$\mathbf{x}=\mathbf{1}$	0101	0011
$\mathbf{x}=\mathbf{0}$	1010	1100

FA or SCA. There are several ways to choose a dynamic encoding with four 4-bit codes. We denote as *m-of-n* encoding an encoding with *m* bits of the same value within a given code of length *n*. A *1-of-4* dynamic encoding would be, for example, $\{0001, 0010, 0100, 1000\}$, and a *2-of-4* one would be realized with four codes within $\{0011, 1100, 1001, 0110, 1010, 0101\}$.

Three different examples of dynamic encoding are represented in the Tables Ia to Ic, referred to as *encoding tables*. Each input bit is represented by two codes, and a control signal c is used to alternate between the two columns of an encoding table. This control signal c is generated with a flip-flop and an inverter, and its value changes at every clock cycle. Therefore, incoming bits are encoded differently at times T and T+1 (modulo 2).

As a consequence, the encoding function $\mathcal{F}$ takes two arguments, x and c, to produce a dynamic encoding of x. The decoding function $\mathcal{F}^{-1}$ outputs the same result for both values of c, as the two codes represent the same input bit. However, we choose to keep c as an argument to $\mathcal{F}^{-1}$, in order to improve the fault coverage, as detailed in Section III. As c alternates between "0" and "1" at each clock cycle, it does not have the same value when encoding a bit x with $\mathcal{F}$ in input of a shift register, and when decoding $\mathcal{F}(x)$ at the output with $\mathcal{F}^{-1}$, in case the length of the shift register is odd. Therefore, $\mathcal{F}^{-1}$ takes as an argument c or $\overline{c}$ if this length is even or odd, respectively. To simplify the notation, in the rest of this article, we write the encoding and decoding functions as $\mathcal{F}_c$ and $\mathcal{F}_c^{-1}$, independently of the length of the shift register. In addition, we denote c as the *parity* of a code $\mathcal{F}_c(x)$, meaning that x has been encoded with $c = 0$ or $c = 1$ if the parity of $\mathcal{F}_c(x)$ is "0" or "1", respectively. For example, in TABLE Ia, the code "1000" has a parity of "0" and represents a bit "1".

Many different 1-of-4 and 2-of-4 dynamic encodings exist. Any permutation of the four codes $\{0001, 0010, 0100, 1000\}$ can realize a 1-of-4 encoding, thus there are $4! = 24$ different such encodings. A k-bit shift register protected with a 1-of-4 encoding has a total HW of k and a total HD of $2k$. Choosing four valid codes among the six codes $\{0011, 1100, 1001, 0110, 1010, 0101\}$ for a 2-of-4 dynamic encoding is more complex, as some of these codes are complementary. Two complementary codes have a HD equal to 4, while two non-complementary codes have a HD equal to 2. There are three pairs of complementary codes, therefore, when selecting the four codes to realize a 2-of-4 dynamic encoding, at least two of these codes are complementary. In order to respect Property 2, two complementary codes can only encode both a "0" and a "1" with the same parity, i.e. within the same column of the encoding table. Therefore, as the parity changes at every clock cycle, the HD between two consecutive codes is always equal to 2. In total, there are 144 possible 2-of-4 dynamic encodings that respect these constraints. A k-bit register protected with such a 2-of-4 encoding has a total HW and a total HD both equal to $2k$.

Fig. 1 represents two examples of dynamic encoding applied to a 7-bit shift register with an arbitrary internal state. Fig. 1a represents the application of the 1-of-4 encoding of TABLE Ia, while Fig. 1b represents the application of the 2-of-4 encoding of TABLE Ib. On these figures, the Hamming weight and Hamming distance are constant for each of the protected registers. They are respectively equal to 7 and 14 with the 1-of-4 encoding, and are both equal to 14 with the 2-of-4 encoding.

III. On the security against fault injections

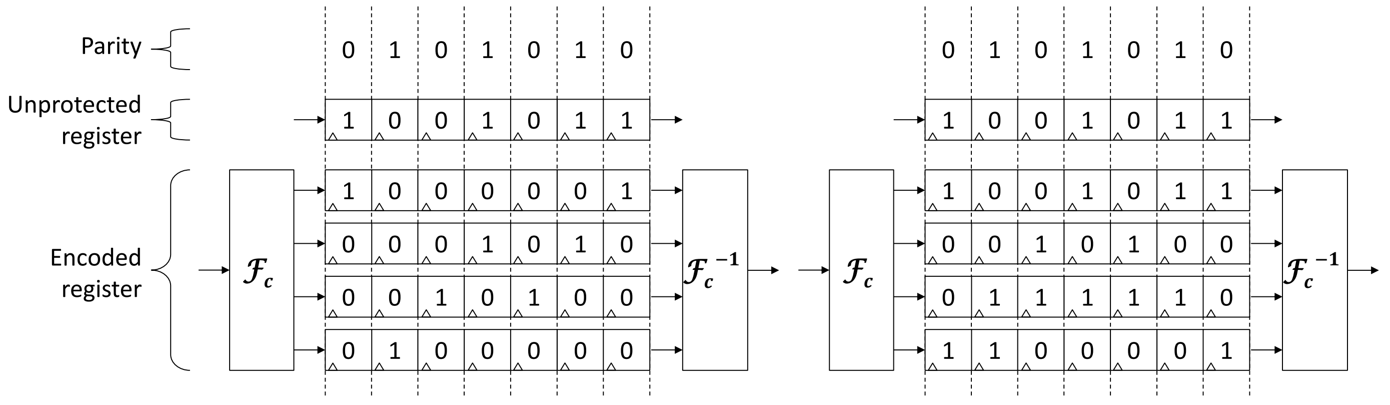
The decoding function $\mathcal{F}_c^{-1}$ that is applied to the output of the protected shift register performs fault detection. Whenever a code cannot be decoded by $\mathcal{F}_c^{-1}$, i.e. if this code is not among the two possible codes for a given parity, this means that this code is faulty and $\mathcal{F}_c^{-1}$ raises an error flag. A fault injection is successful if the induced fault cannot be detected by $\mathcal{F}_c^{-1}$, thus if the fault produces a valid code. In this section, we discuss the security of both 1-of-4 and 2-of-4 dynamic encodings, under several models of faults:

- *Commutation* faults consist in changing the value of a bit, regardless of its initial value, as opposed to the faults consisting in a *reset to 0* or a *set to 1.*
- Faults can be injected over a *single bit* or *several consecutive bits* within one shift register.
- Similarly, faults can be injected over a *single share*, that is, over a single shift register among the four parallel shift register, or over *several shares*, i.e. in several shift register simultaneously.

When the fault injection is successful, the HW of the new code has to be preserved, which means that a commutation from "0" to "1" inside a code has to be balanced by another commutation from "1" to "0", and vice versa. As a consequence, *set* and *reset* faults are automatically detected by $\mathcal{F}_c^{-1}$ and only *commutation* faults can be successful for both 1-of-4 and 2-of-4 encodings. We discuss below the specificities of both types of encodings regarding fault injection.

A. Faults against 1-of-4 dynamic encoding

A successful fault injection should commute *exactly two shares out of four* of a code. The decoding $\mathcal{F}_c^{-1}$ being performed only for a single parity at each clock cycle, these two shares have to be selected carefully. For example, though "1000" is a valid encoding of "1" in TABLE Ia, it is not recognized as such by $\mathcal{F}_c^{-1}$ when c is equal to "1". The



(a) 1-of-4 encoding (b) 2-of-4 encoding

Fig. 1: Examples of an arbitrary shift register protected with a 1-of-4 and a 2-of-4 encoding.

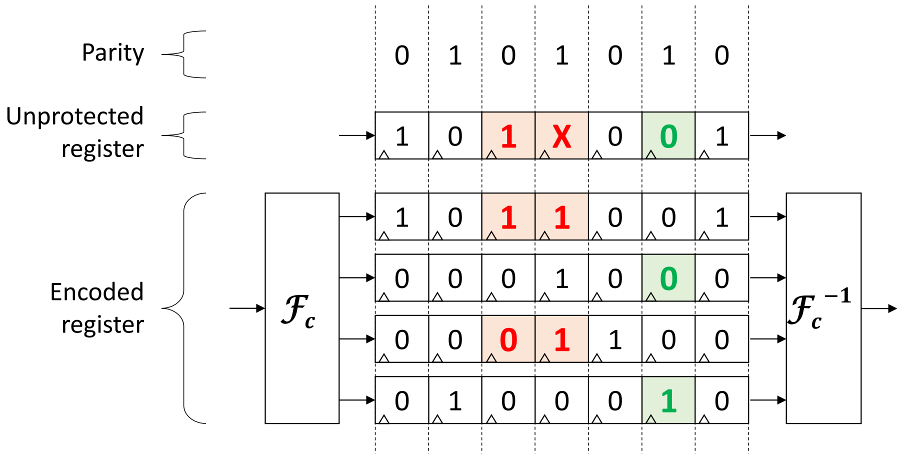


Fig. 2: Examples of faults injected on exactly two shares for a 1-of-4 encoding. A fault targeting exactly one bit in both shares may be undetected (in green), while faults injected on consecutive bits are always detected (in red).

selection of the two shares where the fault has to be injected requires the knowledge of the encoding table, but also of the current value of the control signal c. An example of such a successful fault injection is depicted in green in Fig. 2, based on the same arbitrary register as that depicted in Fig. 1a.

The two shares to target for a successful fault injection are different for two different parities, i.e. the two values of the control signal c. As this parity is different for two consecutive codes, the targeted shares must also be different when fault injection is performed over consecutive bits. As a consequence, faults cannot target several consecutive bits within any share without resulting in invalid codes. An example of fault targeting two consecutive bits is represented in red in Fig. 2. Though this fault is successfully injected in one of these two consecutive encoded bits, it is detected for the other encoded bit.

B. Faults against 2-of-4 dynamic encoding

As stated in Section II, there is at least one pair of complementary codes among the four codes of a 2-of-4 dynamic encoding; these complementary codes encode a “0” and a “1” respectively, for a given parity. The two other codes can be either a complementary pair as well, or not complementary. The former case is illustrated in TABLE Ic, and the latter in TABLE Ib. The fault detection is different depending on these cases, as detailed below.

The encoding table contains two pairs of complementary codes, each for a different parity. As codes are complementary for any parity, injecting a successful fault comes down to inverting any code. In that case, faults can be injected *on all shares*, and on either *one or several consecutive bits* in these shares. The fault detection in that case is less robust than with a 1-of-4 encoding. It is similar to models described in [12]–[14], where only faults injected on at most $(d-1)$ shares out of d are detected.

The encoding table contains only one pair of complementary codes, for a given parity. As previously, faults injected against complementary codes have to be performed *on all shares* of these codes. However, this is only valid for one value of the parity; the two codes are not complementary for the other parity. Fault injection on the two non-complementary codes has to target *exactly two shares out of four*, with the same conditions as with a 1-of-4 encoding. Therefore, two consecutive faults on two consecutive codes have to target alternatively exactly two and four shares. As a consequence, faults against several consecutive bits in a given number of shares are detected.

C. *Conclusion on faults in dynamically encoded shift registers*

An overview of state-of-the-art fault injection techniques is given in [2]. On the one side, a fault on exactly two shares out of four could theoretically be injected with expensive equipment such as two synchronized lasers with small spot sizes. This also requires a good knowledge of both the physical implementation of the countermeasure and the value of the control signal c. On the other side, a fault on all shares could be injected, depending on the physical implementation, with a single laser or EM probe, or with clock or voltage glitches. The attack is facilitated if successful faults can be injected on several consecutive bits within each register, as this requires a lower precision of the injection mechanism. Therefore, the case where two pairs of complementary codes are used to realize a 2-of-4 encoding should be avoided, since the complexity of attacks in that case is much lower than that of attacks against either 1-of-4 encoding or the other type of 2-of-4 encoding.

Most of existing combined protections based on masking [12]–[14] detect only faults injected on at most $(d-1)$ shares out of d. This is also the case for dual-rail logic styles, which are based on a static differential encoding. Similarly, spatial redundancy is vulnerable if a similar fault is injected in all the parallel redundant datapaths. Furthermore, all these countermeasures generally do not add any particular protection against faults injected on several consecutive bits. Based on these observations, dynamic encoding of shift registers provides a better fault coverage than most existing protections, when considering an attack model that is in line with real injection capabilities.

IV. Random dynamic encoding

The dynamic encoding of a shift register consists in balancing the total HD and HW, to balance the theoretical power consumption. However, on a real silicon implementation, there might be a mismatch in the propagation times and capacitances in the four datapaths. These imbalances result in a slightly different power consumption and timing in each datapath, which could be exploited by attackers to perform a successful SCA.

In order to compensate these effects, we propose a *random* dynamic encoding that changes the encoding table at each execution. As stated previously, there are 24 possible 1-of-4 encodings and 144 possible 2-of-4 encodings. At every initialization of the protected shift register, a new encoding table is either generated or chosen among several tables stored in memory, based on a random number. This random number is 5-bit long when choosing among only 1-of-4 encodings, and up to 8-bit long for all possible encodings. By randomly choosing the encoding table at each initialization of the

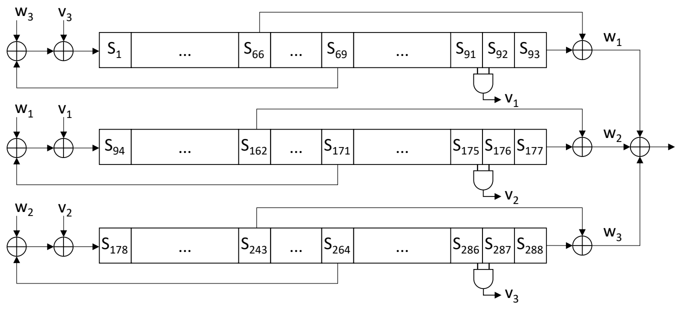


Fig. 3: Unprotected stream cipher Trivium.

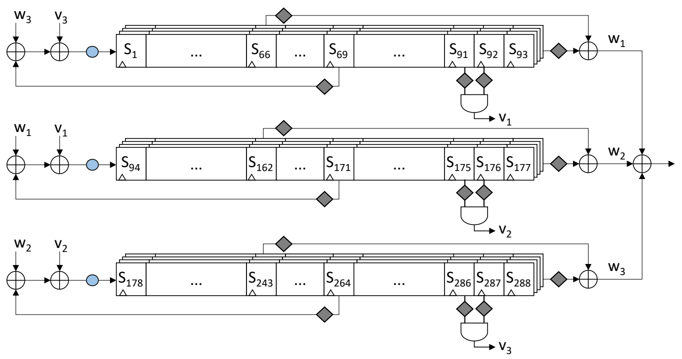


Fig. 4: Dynamic encoding applied to Trivium. Blue circles represent the encoding of the register inputs with $\mathcal{F}_c$, and grey diamonds represent the decoding of register outputs with $\mathcal{F}_c^{-1}$.

protected shift register, the remaining side-channel leakage is randomized and is thus harder to exploit by attackers.

In addition, the complexity of fault injections further increases when encoding tables are generated randomly at each initialization of the protected register. In that case, the encoding table is not known to an attacker, and thus the choice of which shares to target for fault attacks has to be performed randomly. As the encoding table changes at each execution, the reproducibility of a successful fault injection is reduced.

V. Implementation of random dynamic encoding

We apply random dynamic encoding to two use cases, being respectively a 128-bit shift register and the stream cipher Trivium [16]. Both protected implementations can use any of the 24 possible 1-of-4 encodings at each initialization. In the first case, we assess the implementation cost and security of this countermeasure when it is applied only to sequential logic. In order to obtain the implementation cost and side-channel leakage of protected sequential logic only, the encoding and decoding functions $\mathcal{F}_c$ and $\mathcal{F}_c^{-1}$ are excluded from the analysis on the protected shift register. In the second case, we evaluate the cost and efficiency of random dynamic encoding in a representative instance of a lightweight security primitive, which is composed of both sequential and combinational logic. In that case, the additional logic that performs the encoding and decoding, that randomly generates encoding tables, and that encodes the initial internal state of Trivium, is included in both the cost and the security analysis.

TABLE II: Implementation results

Design	Cell area (GE)	Average power (μW)
R_UNPROTECTED	986	103
R_RAND_ENC	3930	414
T_UNPROTECTED	2808	241
T_RAND_ENC	12764	1024

Trivium is a lightweight hardware-oriented stream cipher adapted for resource-constrained applications in the IoT. It has an internal state of 288 bits contained within three non-linear feedback shift registers and is initialized with a secret key and a known initialization vector (IV) that are both 80-bit long. A schematic of an unprotected Trivium is shown in Fig. 3. In the dynamically encoded Trivium, the encoding function $\mathcal{F}_c$ is applied to the input of each of the three encoded shift registers, and the decoding function is applied to every output of these encoded registers, which is represented in Fig. 4. The additional logic to encode the key and IV before they are loaded, as well as the logic necessary to generate a random encoding table at each initialization of the protected Trivium, is not represented in this figure.

In the following, we call *R_UNPROTECTED* and *T_UNPROTECTED* the unprotected implementations of respectively the shift register and Trivium, while *R_RAND_ENC* and *T_RAND_ENC* denote the implementations protected by random dynamic encoding.

We perform the synthesis of these designs with Synopsys Design Compiler and the layout with Cadence Innovus. We use the 28 nm FDSOI library from STMicroelectronics, with a typical Process-Voltage-Temperature corner at an operating voltage of 0.9 V and a frequency of 330 MHz. The average power consumption of each implementation is obtained with 10 back-annotated post-layout simulations over 100 clock cycles with different keys and IVs for Trivium, and with different initial values for the shift register. The implementation results are reported in TABLE II. As the protected shift register has four times more flip-flops than the unprotected one, its surface and power consumption are also four times higher. The overhead is slightly higher than 4 when protecting Trivium, with an increase by 4.55 times of the surface, and by 4.25 of the power consumption. This additional implementation cost comes from the encoding and decoding functions, the random generation of encoding tables at each initialization, and the loading of encoded key and IV at initialization.

Note that unlike most combined countermeasures based on masking, random dynamic encoding does not introduce additional clock cycles during the encryption. Therefore, its energy overhead is equal to its average power overhead, and the energy consumption of a protected Trivium is 4.25 times that of an unprotected implementation.

VI. Resistance against side-channel analysis

The resistance to SCA is evaluated with back-annotated post-layout simulations. These simulations do not include any kind of noise, which represents an ideal case for attackers.

TABLE III: Maximum absolute values of the NED, NSD, and correlation coefficient

Design	NED	NSD	Correlation
R_UNPROTECTED	0.2272	0.0381	0.9989
R_RAND_ENC	0.0	0.0	0.0073
T_UNPROTECTED	0.1705	0.0271	0.8950
T_RAND_ENC	0.0203	0.0032	0.0035

This analysis is performed for the four implementations described in Section V. For all evaluated designs, no specific constraint is applied to the placement or routing during the physical implementation and thus datapaths are imbalanced. This reflects an implementation of dynamic encoding at the algorithmic level only, with a standard implementation flow.

A. Energy per clock cycle

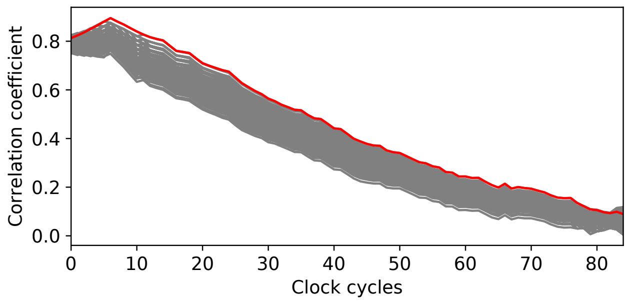
We evaluate the variations of the energy per clock cycle to assess the efficiency of power balancing. For all designs, 1000 simulated power traces, each over 100 clock cycles, are obtained with different initialization parameters. For each clock cycle, we evaluate the Normalized Energy Deviation (NED) and the Normalized Standard Deviation (NSD) over the 1000 traces. The closer these metrics are to 0, the more balanced is the energy consumption. The maximum NED and NSD, taken over all clock cycles, are given in TABLE III.

In the case of Trivium, applying dynamic encoding reduces the NED and NSD by respectively 8.4 and 8.5. The energy per clock cycle of the protected shift register is constant, and thus both the NED ans NSD are null in that case. This indicates that this countermeasure efficiently balances the energy consumption of sequential logic, without even requiring a specific placement or routing.

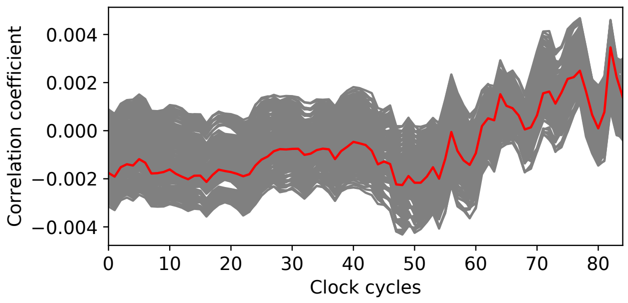
B. Correlation coefficient

The correlation coefficient is used when performing a Correlation Power Analysis (CPA). Unlike the NED and NSD, the correlation is calculated with the instantaneous power consumption at each sample point. Therefore, it captures more accurately the side-channel leakage in short power peaks and glitches and assesses the vulnerability of a design to a practical CPA. For each design, 100,000 power traces with different initialization parameters are simulated. These traces cover the first 85 clock cycles of the execution of Trivium, since published side-channel attacks against Trivium require the measurement of side-channel information during 78 to 81 clock cycles [5]–[8]. The correlation is calculated for the first byte of the key-dependent intermediate values, as described in [8]. For the shift registers, it is calculated for the first 8 input bits, over the first 85 clock cycles as well. For all implementations, the Hamming distance is used as the power model.

The maximum absolute values of the correlation coefficients over the whole length of the traces are displayed in TABLE III. While correlations for both unprotected implementations are high, we observe a significant reduction of the maximum correlation coefficient for both protected implementations. In



(a) T_UNPROTECTED



(b) T_RAND_ENC

Fig. 5: Correlation coefficient over 85 clock cycles, for a byte of key. The correct key hypothesis is displayed in red.

addition, Fig. 5 represents the evolution of the correlation coefficient for the 256 possible hypotheses for the targeted byte of the key, the correct hypothesis being displayed in red. The correlation for the correct key hypothesis cannot be distinguished from other key hypotheses in the case of the protected Trivium. This denotes an absence of correlation for protected implementations, and thus the impossibility to perform a CPA with the existing power model.

C. *Leakage assessment with Welch's T-test*

The correlation coefficient only evaluates the possibility to perform existing state-of-the-art attacks, with a given power model and specific intermediate values. In addition, we perform a leakage assessment with a non-specific Welch's T-test, which does not depend on particular intermediate values or leakage models. It assesses general information leakage, regardless whether this leakage is practically exploitable or not. This test evaluates whether two sets of power traces come from the same distribution; if they do not, it means that these sets are distinguishable, and thus that there is information leakage. The non-specific T-test, also called fixed-versus-random T-test, consists in creating these two sets of traces the following way: one set is generated with fixed parameters, e.g. the key and IV in the case of Trivium, and the other is generated with random parameters, e.g. random IVs for Trivium. Each of these datasets contains 100,000 simulated power traces, over 85 clock cycles. According to [20], whenever the T-test value goes beyond the threshold value of ± 4.5, there is information leakage with a confidence higher than 0.99999.

The results of these tests are represented in Fig. 6. These results indicate a high leakage for both unprotected implementations. There is no information leakage for the shift register protected by random dynamic encoding. This leakage is highly reduced when this countermeasure is applied to the sequential logic of Trivium, even though the combinational logic is not protected at all in that case. A small side-channel leakage is observed after 40 clock cycles, which might be caused by this unprotected combinational logic. As published side-channel attacks against Trivium are based on the analysis of information leakage during the first clock cycles of the initialization, this countermeasure protects efficiently the implementation of Trivium against existing attack models.

VII. Conclusion

We presented *dynamic encoding*, a new combined countermeasure that protects the implementation of security primitives against both SCA and FA. It provides a better fault coverage than most existing combined countermeasures, and simulations indicate a good resistance against side-channel analysis. Dynamic encoding is particularly efficient when encoding tables are generated randomly at every execution. Applied to Trivium, random dynamic encoding has area and power overhead factors of 4.55 and 4.25 respectively, does not introduce delays during the encryption, and requires at most 8 bits of randomness at every execution of the algorithm. Therefore, it is less expensive than other existing combined countermeasures. Future work includes the study of the application of this countermeasure to combinational logic, in order to protect other lightweight security primitives such as PRESENT or ARX-based designs. In addition, this countermeasure will be evaluated on an ASIC to assess its resilience against side-channel attacks in presence of noise, and the practical detection of faults.

References

[1] "Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016," 2017, https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016 (Accessed: 2020-04-04).

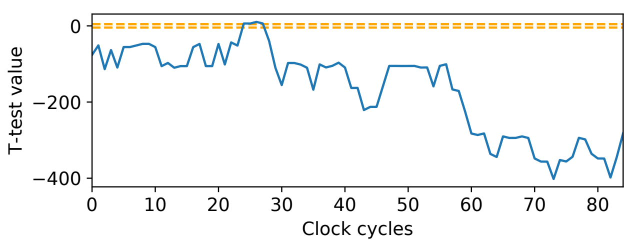
[2] N. F. Galathy, B. Yuce, and P. Schaumont, "A Systematic Approach to Fault Attack Resistant Design," in *Fundamentals of IP and SoC Security: Design, Verification, and Debug*, S. Bhunia, S. Ray, and S. Sur-Kolay, Eds. Cham: Springer International Publishing, 2017, pp. 223–245.

[3] L. Song and L. Hu, "Improved Algebraic and Differential Fault Attacks on the KATAN Block Cipher," in *ISPEC 2013*, ser. LNCS, R. H. Deng and T. Feng, Eds. Springer Berlin Heidelberg, 2013, pp. 372–386.

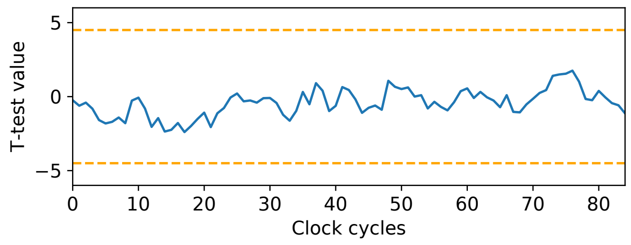
[4] F. E. Potestad-Ordóñez, C. J. Jiménez-Fernández, and M. Valencia-Barrero, "Vulnerability Analysis of Trivium FPGA Implementations," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 12, pp. 3380–3389, Dec. 2017.

[5] W. Fischer, B. M. Gammel, O. Kniffler, and J. Velten, "Differential Power Analysis of Stream Ciphers," in *Topics in Cryptology – CT-RSA 2007*, ser. LNCS. Springer, Berlin, Heidelberg, Feb. 2007, pp. 257–270.

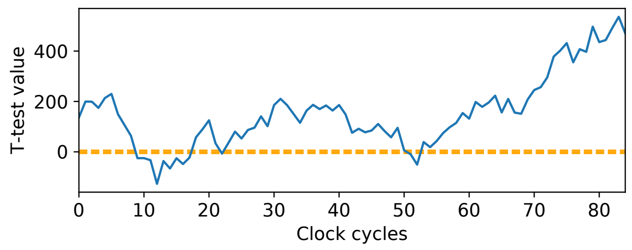
[6] D. Strobel, "Side Channel Analysis Attacks on Stream Ciphers," Ph.D. dissertation, Ruhr-Universität Bochum, 2009, https://www.researchgate.net/publication/251447277_Side_Channel_Analysis_Attacks_on_Stream_Ciphers (Accessed: 2020-04-04).



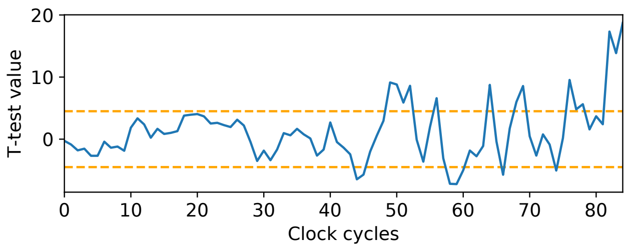
(a) R_UNPROTECTED



(b) R_RAND_ENC



(c) T_UNPROTECTED



(d) T_RAND_ENC

Fig. 6: Fixed-vs-random T-tests

[7] G. V. Bard, N. T. Courtois, J. Nakahara, P. Sepehrdad, and B. Zhang, "Algebraic, AIDA/Cube and Side Channel Analysis of KATAN Family of Block Ciphers," in *Progress in Cryptology - INDOCRYPT 2010*, ser. LNCS, G. Gong and K. C. Gupta, Eds. Springer Berlin Heidelberg, 2010, pp. 176–196.

[8] Y. Jia, Y. Hu, F. Wang, and H. Wang, "Correlation power analysis of Trivium," *Security and Communication Networks*, vol. 5, no. 5, pp. 479–484, 2011.

[9] P. Maistri, "Countermeasures against fault attacks: The good, the bad, and the ugly," in *2011 IEEE 17th International On-Line Testing Symposium*, Jul. 2011, pp. 134–137.

[10] K. Tiri and I. Verbauwhede, "A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation," in *Proceedings of the Conference on Design, Automation and Test in Europe*, vol. 1, Feb. 2004, pp. 246–251.

[11] T. Schneider, A. Moradi, and T. Güneysu, "ParTI – Towards Combined Hardware Countermeasures Against Side-Channel and Fault-Injection Attacks," in *Advances in Cryptology – CRYPTO 2016*, ser. LNCS, M. Robshaw and J. Katz, Eds. Springer Berlin Heidelberg, 2016, pp. 302–332.

[12] T. D. Cnudde and S. Nikova, "Securing the PRESENT Block Cipher Against Combined Side-Channel Analysis and Fault Attacks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 12, pp. 3291–3301, Dec. 2017.

[13] O. Reparaz, L. De Meyer, B. Bilgin, V. Arribas, S. Nikova, V. Nikov, and N. Smart, "CAPA: The Spirit of Beaver Against Physical Attacks," in *Advances in Cryptology – CRYPTO 2018*, ser. LNCS, H. Shacham and A. Boldyreva, Eds. Springer International Publishing, 2018, pp. 121–151.

[14] L. D. Meyer, V. Arribas, S. Nikova, V. Nikov, and V. Rijmen, "M&M: Masks and Macs against Physical Attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 25–50, 2019.

[15] T. D. Cnudde, O. Reparaz, B. Bilgin, S. Nikova, V. Nikov, and V. Rijmen, "Masking AES with d+1 Shares in Hardware," in *CHES 2016*, ser. LNCS. Springer, Berlin, Heidelberg, 2016, pp. 194–212.

[16] C. D. Canniere and B. Preneel, "TRIVIUM Specifications," *ECRYPT Stream Cipher Project*, 2006, https://www.ecrypt.eu.org/stream/e2-trivium.html (Accessed: 2020-04-04).

[17] "The eSTREAM portfolio page," http://www.ecrypt.eu.org/stream (Accessed: 2020-04-04).

[18] C. De Cannière, O. Dunkelman, and M. Knežević, "KATAN and KTANTAN — A Family of Small and Efficient Hardware-Oriented Block Ciphers," in *CHES 2009*, ser. LNCS, C. Clavier and K. Gaj, Eds. Springer Berlin Heidelberg, 2009, pp. 272–288.

[19] J.-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia, "Quark: A Lightweight Hash," in *CHES 2010*, ser. LNCS, S. Mangard and F.-X. Standaert, Eds. Springer Berlin Heidelberg, 2010, pp. 1–15.

[20] T. Schneider and A. Moradi, "Leakage Assessment Methodology," in *CHES 2015*, ser. LNCS. Springer, Berlin, Heidelberg, Sep. 2015, pp. 495–513.