IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2019, No. 1, pp. 5–45. DOI:10.13154/tosc.v2019.i1.5-45

CRAFT: Lightweight Tweakable Block Cipher with Efficient Protection Against DFA Attacks

Christof Beierle[1], Gregor Leander[2], Amir Moradi[2] and Shahram Rasoolzadeh[2]

[1] Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, Luxembourg, Luxembourg
beierle.christof@gmail.com
[2] Horst Görtz Institute for IT Security, Ruhr-Universität Bochum, Bochum, Germany
firstname.lastname@rub.de

Abstract. Traditionally, countermeasures against physical attacks are integrated into the implementation of cryptographic primitives after the algorithms have been designed for achieving a certain level of cryptanalytic security. This picture has been changed by the introduction of `PICARO`, `ZORRO`, and `FIDES`, where efficient protection against Side-Channel Analysis (SCA) attacks has been considered in their design. In this work we present the tweakable block cipher `CRAFT`: the efficient protection of its implementations against Differential Fault Analysis (DFA) attacks has been one of the main design criteria, while we provide strong bounds for its security in the related-tweak model. Considering the area footprint of round-based hardware implementations, `CRAFT` outperforms the other lightweight ciphers with the same state and key size. This holds not only for unprotected implementations but also when fault-detection facilities, side-channel protection, and their combination are integrated into the implementation. In addition to supporting a 64-bit tweak, `CRAFT` has the additional property that the circuit realizing the encryption can support the decryption functionality as well with very little area overhead.

Keywords: `CRAFT` · block cipher · tweakable · lightweight · fault detection · involutory

1 Introduction

After almost two decades of the introduction of *physical attacks* [16, 57, 58], it is widely known that the secrets stored in and processed by an implementation of strong cryptographic algorithms can be recovered by means of physical attacks. One of the most powerful class of such threats is certainly fault-injection attacks [16], where the adversary disturbs the cryptographic device during its operation. Such disturbances, which are usually transient faults, can be created by means of a clock glitch [3] (which violates the delay of the circuit's critical path), under-powering [23, 79] (which, in addition to setup-time violation, may modify the circuit's execution flow), an EM glitch [31] (which can change the transistors' state), or a laser beam [2, 23] (which as the most precise mean can change the state of particular transistors). Their feasibility is mainly tied with the fact that the attacker – in many applications such as pay-TV or electronic money – is actually a legitimate user. Thus, we face the situation where the cryptographic devices are in the hands of the adversary.

As a result, integrating countermeasures to prevent such physical attacks in general and fault attacks in particular is essential for products that offer security and privacy. As an example, for many years smart card (e.g. bank card) manufacturers had to[1] integrate such

[1]It is forced if particular certification is needed, e.g. common criteria evaluation.

Received: 2018-09-01, Revised: 2018-11-23, Accepted: 2019-01-23, Published: 2019-03-08

techniques at different levels of abstraction. However, such countermeasures usually come with a significant cost. Since the cryptographic algorithms are usually designed considering their robustness against cryptanalytic attacks, the integration of fault detection schemes into their implementation becomes – most of the times – challenging and not necessarily efficient. Indeed, integrating countermeasures easily increases the implementation costs usually by a factor of at least two (see Section 2).

In this work we focus on countermeasures against Differential Fault Analysis (DFA) attacks which are defined at algorithmic level with the area cost as the performance parameter. More precisely – instead of hindering the faults – we focus on schemes that try to detect the faults during the computations[2]. Generally speaking, the algorithmic-level countermeasures to fault-injection attacks have to add redundancy to the implementation to enable examining the consistency of the performed operations, hence fault detection. Trivial examples include *timing redundancy* [63, 64] (e.g. by repeating the operations) and *area redundancy* [46, 64] (e.g. by re-instantiating equal modules which perform the same operations)[3]. Since the consistency of information is checked simultaneously with the computation, such schemes are usually denoted as Concurrent Error Detection (CED).

Most of fault-detection mechanisms follow the reliability concept, i.e., with the goal of increasing the percentage of detectable faults, which may occur due to environmental effects. However, resistance against fault-injection attacks is based on a different concept, where protection against an adversary who has certain bounded abilities should be achieved. Recently, a mechanism has been introduced in [1] which can guarantee the detection of faults injected by an attacker with the ability of making a bounded number of cells in the entire circuit faulty. The underlying approach is a CED scheme constructed over an Error Detecting Code (EDC) which can be easily adjusted by increasing the minimum distance of the code, i.e., the maximum number of faults that the code can detect + 1. The authors highlighted the *fault propagation* effect and introduced the *independence* property to be fulfilled as a requirement in order to guarantee the detection of up to $t = d - 1$ faults, when d is the minimum distance of the underlying EDC. The authors of [1] have applied their proposed scheme on several different lightweight ciphers and compared their area overhead.

Ineffective Fault Attacks. Compared to DFA [16], other attack vectors like safe-error [89] and Ineffective Fault Attacks (IFA) [26] exploit the secret by just examining whether the output is faulty or not. The same concept is followed in Fault Sensitivity Analysis (FSA) [60] as well. While safe-error attacks are mainly applied on asymmetric cryptography, IFA usually makes use of a precise fault model (e.g. stuck-at-0) which necessitate sophisticated fault injection tools like laser beams, particularly challenging when modern nano-scale circuits are targeted. In contrary a clock glitch is used in FSA to violate the critical-path delay of the circuit for a subset of the given inputs. FSA exploits the time required by a combinatorial circuit dealing with a secret, e.g. a realization of an Sbox. Then, FSA conducts the attack by means of a hypothetical model similar to Correlation Power Analysis (CPA) [22].

In the seminal work [34] the Statistical Ineffective Fault Attack (SIFA) has been introduced that relaxes the necessity of a precise fault model of IFA, and at the same time generalizes FSA by not requiring any hypothetical model. It is indeed able to break many implementations protected by countermeasures against fault attacks. Its effectiveness even on masked implementations is recently shown in [33].

The fault-detection mechanism which we consider in our designs cannot by itself counteract IFA, FSA or SIFA. As stated in [34], detection-based countermeasures need to be combined with other types of countermeasures to provide security against SIFA.

[2]Once a fault is detected, the operation can either be stopped or continued with random data [39, 90].
[3]For a detailed survey, the interested reader is referred to [43].

There are two possible generic ways to counter SIFA. First, instead of detecting errors only, error-correction facilities would harden the implementations against SIFA. Second, mechanisms to limit the number of faulty executions, e.g. implemented by a counter that counts the number of detected errors and shuts down the device permanently after a certain number of faults have been detected, provide a conceptual simple way to lift detection-based countermeasures to counteract SIFA. Here in this work, we deal only with the detection-based part of such a combination. Moreover, we exclude the safe-error and stuck-at faults in our adversary model.

1.1 Lightweight Cryptography

In symmetric cryptography, motivated by new application scenarios – in particular the Internet of Things – *lightweight* cryptography has been a very active research area in the last decade. While there is no strict definition of the term lightweight cryptography, it is usually understood as cryptography with a strong focus on efficiency. Here efficiency can be measured according to various criteria and their combination.

The first generation of lightweight ciphers, e.g. `PRESENT` [19] and `KATAN` [24], focused on chip area only and used very simple round functions as the main building block. Later generations of lightweight ciphers broadened the scope significantly. By now, we have at hand dedicated ciphers optimized with respect to code-size (e.g. `PRIDE` [4] and `SPECK` [10]), latency (e.g. `PRINCE` [21], `MANTIS` [12] and `QARMA` [6]), efficiency of adding countermeasures against passive Side-Channel Analysis (SCA) attacks (e.g. the family of LS-Designs [40], `PICARO` [71] and `ZORRO` [38] (software oriented), `FIDES` [17] and actually `NOEKEON` [30] (hardware oriented) even before the term lightweight cryptography was invented), efficient fault detection (e.g. `FRIT` [81]), and energy (e.g. `MIDORI` [7] and `GIFT` [8]). Moreover, the overhead of implementing decryption on top of encryption has been subject to optimization (e.g. `ICEBERG` [82], `MIDORI` and `NOEKEON` where the components are involutions and `PRINCE` with its α-reflection property).

Besides focusing on various criteria, there has also been an advance in the general design philosophy. Indeed, many recent lightweight ciphers (e.g. `LED` [41], `SKINNY` [12] and `MIDORI`) use the general framework of the AES round function and fine-tune its components to achieve better performance while previous constructions were rather ad-hoc. Borrowing from AES in particular allows for simpler security analysis, following e.g. the wide-trail strategy (see [28]). More recently, there are also attempts to design lightweight tweakable block ciphers, a block cipher that is extended with an additional public input, the tweak; this primitive allows for better encryption modes and efficient constructions of authenticated encryption schemes [61]. Examples here include `SKINNY`, `MANTIS`, `QARMA`, and `Joltik`, and the TWEAKEY framework [45] as a general design principle.

1.2 Our Contribution

In this work, we intend to construct a new symmetric cipher for which protection against DFA attacks has been considered in its design phase. To this end, we first show that any cipher that allows the fault-detection unit to solely operate on the redundant part of information with strictly shorter size than that of the plaintext/ciphertext has a critical cryptographic weakness. In the second part, we introduce the tweakable block cipher `CRAFT` (with effiCient pRotection Against differential Fault analysis aTtacks) with 64-bit plaintext/ciphertext and 128-bit key width. In short, its properties are listed below.

- In addition to having strong cryptographic properties, `CRAFT` has been particularly designed to ease the integration of code-based fault-detection schemes following the concept presented in [1]. This allows the application of any arbitrary EDC in the implementation; we have considered four such codes in our case studies. Recently, the permutation FRIT [81] was introduced in which efficient implementation of a fault-detection technique has been considered as a design criteria. Although broken [35], FRIT uses *interleaved parity* for fault detection which – based on the definition of the underlying code – can guarantee the detection of only single-bit faults.

- Due to the involutory property of its fundamental building blocks, the CRAFT encryption function can easily be turned to decryption, supporting both encryption and decryption with minimal area cost.
- CRAFT supports a 64-bit tweak, which also adds a very little area overhead to the corresponding implementation.
- Considering the area footprint of a round-based architecture, where each round of the cipher is computed in one clock cycle, CRAFT outperforms – to the best of our knowledge – all lightweight block ciphers with the same state and key size. As a remarkable outcome, its encryption-only core needs 949 GE, which is way lower than any reported round-based implementation of a lightweight cipher[4]. It indeed competes with a bit-serial implementation of SIMON with 958 GE requiring 64×44 clock cycles [10]. Among the key features that enable these achievements is the construction of the key schedule in CRAFT, where – similar to MIDORI, PICCOLO [80], and KTANTAN [24] – the key bits are alternated. This allows us to avoid instantiating extra registers to process and generate the round keys in round-based architectures.
- Focusing on fault-protected implementations, under all settings with respect to the employed EDC, its area overhead (even with decryption and tweak support) is smaller than all block ciphers considered in [1] with compatible state and key size (see Table 6).

Clearly, we build upon the knowledge and experience that has been established for building lightweight (tweakable) ciphers by now. This is reflected in the fact that CRAFT, on a high level view, is quite similar to, e.g. SKINNY, which itself borrows the general structure of the AES. As mentioned above, this in particularly allows us to base our security analyses on well-known principles.

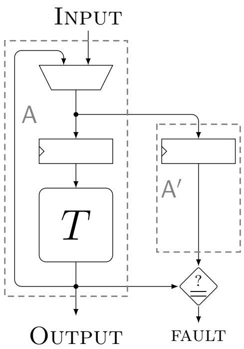
2 On Redundancy

The concept of counteracting fault-injection attacks has sometimes been confused with a fault-tolerance design methodology. As a result, the dependability community has introduced several fault-detection schemes for cryptographic hardware with marginal capability to counteract fault-injection attacks [13, 50, 51, 52, 53, 87]. The main reason behind such shortcomings is the difference between the nature of the faults in these two concepts. The dependability community consider the faults as single-event-upsets (SEUs) that may rarely happen during the operation of the system and that are of limited weight, e.g. a few bits in the entire circuit. In the case of DFA, depending on the underlying fault model the adversary tries to hit several cells of the circuit as many times as he needs to recover the secret. From another point of view, the trade-off between the countermeasure's overhead and the fault coverage[5] plays an important role for the dependability community while insuring the detection of all possible faults under a certain adversary model is of crucial importance for the physical security community.

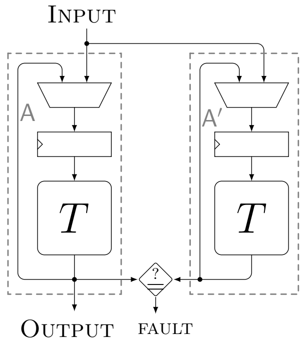
In order to detect any kind of fault, the circuit needs to be equipped with a type of redundancy, i.e., a facility which enables examining the correctness of a computation that

[4] We exclude KTANTAN due to its 80-bit key size and high number of clock cycles for an encryption.

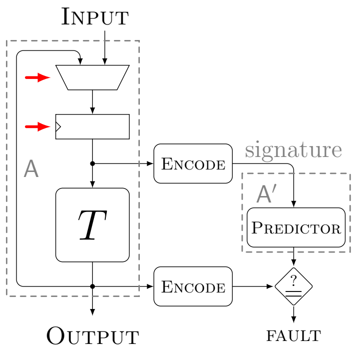
[5] Fault coverage is calculated by $\sum_{\mathrm{d}} / \sum_{\forall}$ with $\sum_{\mathrm{d}}$ the number of detectable and $\sum_{\forall}$ the number of possible faults.



(a) timing redundancy



(b) area redundancy



(c) information redundancy

Figure 1: Common CED schemes.

is called Concurrent Error Detection (CED). Such a redundancy can be categorized into different classes [43] listed below. Note that in Figure 1, where a block diagram for the corresponding schemes are shown, we consider the underlying cryptographic algorithm as iterations of a round function. The original and redundant parts of implementations are marked by A and A' respectively.

- In **timing redundancy** (see Figure 1(a)), the computation is repeated one (or more) time(s) by the same piece of hardware, and the results are compared [64]. As the name says, it has a timing overhead, which linearly increases by the number of times a computation is repeated. In spite of its simplicity, it cannot detect permanent faults.
- In **area redundancy** (see Figure 1(b)), the target module is instantiated more than once in such a way that they all operate in parallel with the same input, and their results can be compared. Although it has no timing overhead, it obviously leads to two (or more) times area overhead. However, it can detect both transient and permanent faults. The simplest version of such a scheme is known as **duplication**, where the entire circuit is instantiated two times [64].
- In **information redundancy** (see Figure 1(c)), the goal is to keep the null timing overhead, but to reduce the area overhead compared to duplication. Instead of fully instantiating the same module twice, it computes a signature (e.g. parity) that performs a correctness check [13, 52, 53, 87, 47, 51, 50]. Of course, there is a trade-off between the area overhead and the efficiency of a fault-detection scheme. In the most extreme case when the redundancy is a single parity bit, the area overhead is minimized, but only certain faults can be detected.
- **Hybrid redundancy** is a customized type of either one or a combination of above-explained schemes. For instance, instead of duplication one can compute the inverse of the operations by the second module [48, 77], which leads to both area and limited timing redundancy. As another example known as invariance-base CED, we can refer to [42] which compared to classical timing redundancy can detect permanent faults with a marginal area overhead.

Due to their limited area overhead, the schemes based on information redundancy have been investigated more widely. However, they usually come with serious shortcomings, which is due to the non-transparency of the signature (e.g. parity) to the underlying function. For clarification, consider Figure 1(c); depending on the underlying signature scheme, a certain type of faults can be detected during the computation of the function T. However, it cannot detect any fault injected in the register cells or in the initial multiplexer (marked by red arrows in Figure 1(c)). Considering limited environmental faults, such constructions might be adequate to satisfy the desired fault-tolerance property, but the level of protection that they can provide against fault-injection attacks is very limited. At this point the difference between the fault-tolerance concept and fault-attack resistance becomes clear.

In order to avoid such issues, duplication is a natural and straightforward solution. It can indeed be seen as an information redundancy scheme where the signature is the same as the original data. In spite of its simplicity, duplication has a pitfall of not being able to detect symmetric faults, i.e., those faults which are similarly injected into both original and redundant modules (usually possible by employing two precisely-localized laser beams). Alternatively, a technique has recently been introduced in [1] which can guarantee the detection of up to a certain number of faults in the entire circuit including the data processing, control logic and the consistency check modules. Two constructions for fault detection have been proposed in [1] which are shown in Figure 2. The one in Figure 2(a) is always possible when the size of redundancy is at least as large as that of the original data. Otherwise, the construction in Figure 2(b) is possible which needs a link from the original circuit A. Below, we shortly restate the relevant definitions.

Definition 1. A *binary linear code* $\mathbf{C}$ comprising codewords c with length l, dimension k and minimum distance d is denoted by $[l, k, d]$. The generator matrix G of size $k \times l$ maps a message $x \in \mathbb{F}_2^k$ into the corresponding codeword $c \in \mathbb{F}_2^k$ with $c = x \cdot G$. The minimum distance d is defined as

$$\min \left(\{ w\left(c_1 + c_2\right) | c_1, c_2 \in \mathbf{C}, c_1 \neq c_2 \} \right),$$

with $w(c)$ the number of 1's in the binary representation of c and $+$ denoting the addition in $\mathbb{F}_2$.

In such a setting, if faults are modeled by an additive error vector e (i.e., the faulty codeword c' can be written as $c + e$), a fault is definitely detected if $w(e) < d$. In other words, a code $\mathbf{C}$ with minimum distance d guarantees the detection of up to $d - 1$ bit faults.

Definition 2. The $k \times l$ generator matrix G of a *systematic code* $\mathbf{C}$ can be represented by $G = [I_k|P]$, with I_k the identity matrix of size k. This enables us to write each codeword c as $[x|p]$, i.e., the message padded with a redundant part p generated by the right part of the generator matrix, i.e., P. The redundant part p is of size $m = l - k$ bits and the matrix P has dimension of $k \times m$.

Duplication is indeed a systematic binary linear code with the generator matrix $G = [I_k|I_k]$ which enables representing the codewords as $[x|x]$. Such a code is of minimum distance $d = 2$, that theoretically explains why it cannot detect the symmetric faults $e = [\delta|\delta]$. However, application of other systematic binary linear codes $[l, k, d]$ enables detection of more bits. To this end, two distinct cases have been considered in [1]:

- $m \geq k$. This allows the generator matrix P to be injective. As shown in Figure 2(a) the signatures are obtained as $F : x \mapsto x \cdot P$, which is an injective function. Hence, the redundant module A' can perform the entire computations only on signatures[6] by $T' = F \circ T \circ F^{-1}$.

- $m < k$. In this case, F cannot be injective, and the feasibility of the construction shown in Figure 2(a) is not guaranteed. Therefore, the authors of [1] proposed

[6] In fact, this concept is along the same lines as the dual cipher notion [9].

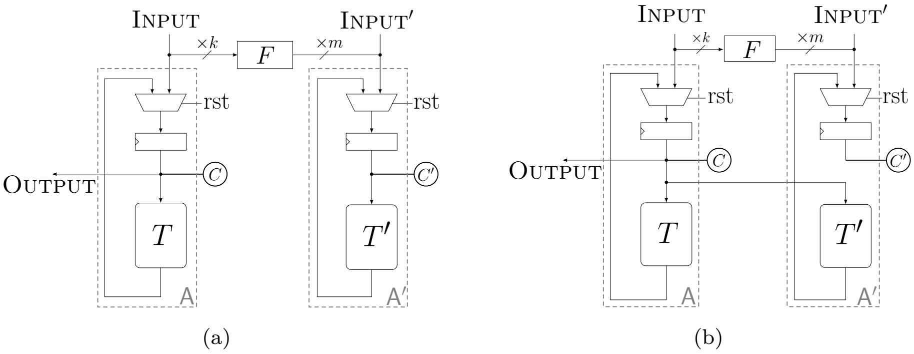


Figure 2: The constructions proposed in [1].

another design shown in Figure 2(b) which forces the redundant module A′ to receive extra information from the original module A (i.e., the input of $T' = F \circ T$ in Figure 2(b) is taken from A).

Note that in both cases shown in Figure 2 the control logic and the consistency check module are not presented. The places marked by Ⓒ and C' are the checkpoints whose consistency should be examined for fault detection. In other words, using extra instances of function F, $c'' = F(c)$ is calculated and $c'' \stackrel{?}{=} c'$ is checked.

It is noteworthy to emphasize that after receiving the signature, A′ in Figure 2(a) operates independently, i.e. without receiving any further information. In contrast, this is not the same case for A′ in Figure 2(b) which requires extra information from A at each clock cycle.

2.1 Lower Bounds for Redundancy

We concentrate on the construction shown in Figure 2(a). In fact, F does not need to be linear. It can be replaced by any injective function and the goal of separation between A and A′ can be fulfilled. The selection of a linear function helps to stay with characteristics of systematic binary linear codes which, compared to non-linear functions, has better fault detection properties[7]. It has another advantage that the algebraic degree of the sub-functions T' stays the same as that of T. This is beneficial when the implementation should also be protected against SCA attacks by Boolean masking [78], e.g. by threshold implementation [68].

It is explained in [1] that if $m < k$, the construction shown in Figure 2(a) is not necessarily feasible. It depends on the employed function F and the underlying computing function T. As a temporary goal, we aim at examining whether it is possible to design a block cipher, with its round function denoted by T, in such a way that its fault-protected implementation (using F) can be realized following the construction in Figure 2(a). In other words, it should be $m < k$, and A′ should solely operate on redundant information (i.e., signature marked as Input′ in Figure 2(a)).

We show here that in the construction of Figure 2(a) if $m < k$, for any function $F : \mathbb{F}_2^k \to \mathbb{F}_2^m$ there is a crucial structural weakness of the underlying cipher as explained in the following.

Theorem 1. *If a cipher can have an error detection structure using area redundancy, where the redundancy part can be processed independently (the structure in Figure 2(a))*

[7]For instance, parity as a linear code can detect all of the single-bit faults while there is no non-linear code with $m = 1$ that can detect all such faults.

with redundancy size of smaller than the block size, then the cipher is not cryptographically secure.

Proof. Let us assume an encryption function with key $\mathbf{k}$, denoted by $\mathcal{E}_{\mathbf{k}}$, formed by repeating the round function T for a certain number of rounds. We also suppose that the bit length of the original data (plaintext/ciphertext) is a multiple of k bits, and the bit length of the redundant information (signature) is a multiple of m bits. In our notation below, by applying the function F we mean its application on each k-bit chunk separately.

By construction, there is a pair of functions (F, F') such that for each instance of a cipher $\mathcal{E}_{\mathbf{k}}$, there exists a block cipher $\mathcal{E}'_{\mathbf{k}}$ such that $\mathcal{E}'_{\mathbf{k}} \circ F = F' \circ \mathcal{E}_{\mathbf{k}}$. Thus, for every two plaintexts $\mathbf{p}_1$ and $\mathbf{p}_2$ for which $F(\mathbf{p}_1) = F(\mathbf{p}_2)$, one obtains $F' \circ \mathcal{E}_{\mathbf{k}}(\mathbf{p}_1) = F' \circ \mathcal{E}_{\mathbf{k}}(\mathbf{p}_2)$. Thus, regardless of the key, whenever two plaintexts have the same image under F, the corresponding ciphertexts will also have the same image under F'. For instance, if (F, F') are balanced functions mapping k to m bits, each of the 2^m different preimages $F^{-1}(x), x \in \mathbb{F}_2^m$ is a set containing 2^{k-m} elements (the same holds for F'^{-1}). Further, every instance $\mathcal{E}_{\mathbf{k}}$ operates as a permutation over these preimages, i.e.,

$$\mathcal{E}_{\mathbf{k}} : F^{-1}(x) \mapsto F'^{-1}(y), \quad \text{if } \mathcal{E}'_{\mathbf{k}} : x \mapsto y.$$

Thus, under a chosen-plaintext attack, the underlying cipher $\mathcal{E}_{\mathbf{k}}$ can easily be distinguished from a random permutation. The adversary chooses plaintexts $\mathbf{p}_1$ and $\mathbf{p}_2$ with $F(\mathbf{p}_1) = F(\mathbf{p}_2)$. If $\mathbf{c}_i = \mathcal{E}_{\mathbf{k}}(\mathbf{p}_i)$, the property $F'(\mathbf{c}_1) = F'(\mathbf{c}_2)$ holds with a probability of 1. Note that this property should hold only with probability 2^{-m} for a uniformly chosen random permutation $\mathcal{E}_{\mathbf{k}} : \mathbf{p}_i \mapsto \mathbf{c}_i$.

□

We conclude that whenever $m < k$, *it is not possible to design a fully secure cipher* if the construction in Figure 2(a) is desired. Therefore, in the rest of the paper we introduce a new cipher with low area overhead considering the construction in Figure 2(a) for $m \geq k$, and the design in Figure 2(b) for $m < k$.

3 Specification of `CRAFT`

`CRAFT` is a lightweight tweakable block cipher made out of involutory building blocks. It consists of a 64-bit block, a 128-bit key, and a 64-bit tweak. The state is viewed as a 4×4 square array of nibbles. We use the notation $I_{i,j}$ to denote the nibble located at row i and column j of the state. One can also view this 4×4 square array as a vector by concatenating the rows. Thus, we denote with a single subscript I_i the nibble in the i-th position of this vector. In other words, $I_{i,j} = I_{4 \cdot i + j}$. Note that all the index counting start from zero.

The 128-bit key K is split into two 64-bit keys K_0 and K_1. Together with the 64-bit tweak input T, four 64-bit tweakeys TK_0, TK_1, TK_2 and TK_3 are derived. Each of those 64-bit tweakeys is also considered as a 4×4 square array of nibbles and we use similar indexing as for the cipher state. By initializing the state with the plaintext, the cipher iterates 31 round functions ($\mathcal{R}_i$, $0 \leq i \leq 30$) and appends one more linear round $\mathcal{R}'_{31}$) to compute the ciphertext. Figure 3 depicts the structure of `CRAFT`. Each round function $\mathcal{R}_i$ applies the following five involutory round operations: `SubBox`, `MixColumn`, `PermuteNibbles`, `AddConstant`$_i$ and `AddTweakey`$_i$, while $\mathcal{R}'_{31}$ only applies the `MixColumn`, `AddConstant`$_i$ and `AddTweakey`$_i$ operations. The round operations are defined as follows, and one full-round function is depicted in Figure 4.

`SubBox (SB):` The 4-bit involutory Sbox S is applied 16 times in parallel, i.e., to each nibble of the state. This Sbox is the same as the Sbox used in the block cipher `MIDORI` [7]. The table for the Sbox (in hexadecimal notation) is given in Table 1.

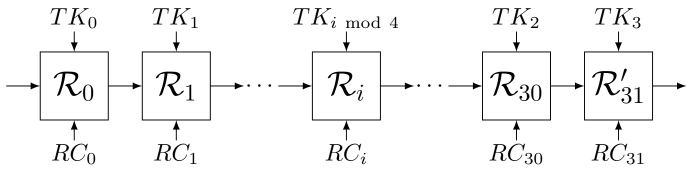


Figure 3: Structure of `CRAFT`

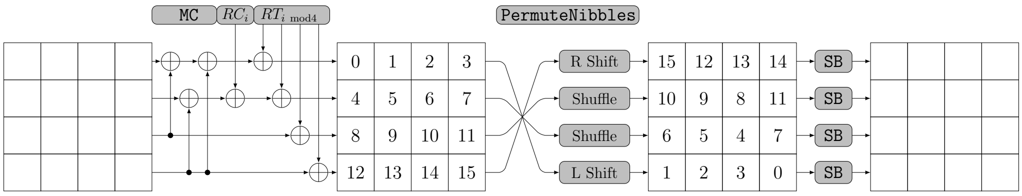


Figure 4: One full round function of `CRAFT`

Table 1: The Sbox of `MIDORI` and `CRAFT`

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x)$	c	a	d	3	e	b	f	7	8	9	1	5	0	2	4	6

`MixColumn (MC):` The following involutory binary matrix M is multiplied to each column of the state:

$$M = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

That is, for each column index $j \in \{0, \ldots, 3\}$,

$$\begin{bmatrix} I_{0,j} \\ I_{1,j} \\ I_{2,j} \\ I_{3,j} \end{bmatrix} \mapsto \begin{bmatrix} I_{0,j} \oplus I_{2,j} \oplus I_{3,j} \\ I_{1,j} \oplus I_{3,j} \\ I_{2,j} \\ I_{3,j} \end{bmatrix} .$$

`PermuteNibbles (PN):` An involutory permutation $\mathcal{P}$ is applied on the nibble positions of the state. In particular, for all $0 \le i \le 15$, I_i is replaced by $I_{\mathcal{P}(i)}$, where

$$\mathcal{P} = [15, 12, 13, 14,\ 10, 9, 8, 11,\ 6, 5, 4, 7,\ 1, 2, 3, 0] .$$

`AddConstants`$_i$ `(ARC`$_i$`):` One 4-bit and one 3-bit LFSR, whose states are denoted by $a = (a_3, a_2, a_1, a_0)$ and $b = (b_2, b_1, b_0)$ (with a_0 and b_0 being the least significant bits), respectively, are used to generate round constants. The LFSRs are initialized by the values (`0001`) and (`001`) and their update functions are

$$(a_3, a_2, a_1, a_0) \to (a_1 \oplus a_0, a_3, a_2, a_1) \quad , \quad (b_2, b_1, b_0) \to (b_1 \oplus b_0, b_2, b_1) .$$

In every round, (a_3, a_2, a_1, a_0) and $(0, b_2, b_1, b_0)$ are XOR-ed with the state nibbles I_4 and I_5, respectively, and then both LFSRs get updated. Table 2 shows the hexadecimal values of all round constants.

Table 2: Round constants of `CRAFT`

Round i	$RC_i = (a, b)$
0 - 15	`11, 84, 42, 25, 96, c7, 63, b1, 54, a2, d5, e6, f7, 73, 31, 14`
16 - 31	`82, 45, 26, 97, c3, 61, b4, 52, a5, d6, e7, f3, 71, 34, 12, 85`

$\texttt{AddTweakey}_i$ ($\texttt{ATK}_i$): Using a permutation Q on the nibbles of the given tweak, the cipher derives four 64-bit tweakeys TK_0, TK_1, TK_2 and TK_3 from the tweak T and the key $(K_0||K_1)$ as

$$TK_0 = K_0 \oplus T\ ,\ TK_1 = K_1 \oplus T\ ,\ TK_2 = K_0 \oplus Q(T)\ ,\ TK_3 = K_1 \oplus Q(T)\ .$$

Thereby, $Q(T)$ applies the permutation

$$\mathcal{Q} = [12, 10, 15, 5,\ \ 14, 8, 9, 2,\ \ 11, 3, 7, 4,\ \ 6, 0, 1, 13]$$

on the nibbles of the tweak T where for all $0 \leq i \leq 15$, T_i is replaced by $T_{\mathcal{Q}(i)}$. Then in each round i, without any key update, the tweakey $TK_{i \bmod 4}$ is XOR-ed to the cipher state.

Round Function: To conclude, using the above explained operations, the round functions $\mathcal{R}_i$, $i \in \{0, \dots, 30\}$, are defined as

$$\mathcal{R}_i = \texttt{SB} \circ \texttt{PN} \circ \texttt{ATK}_i \circ \texttt{ARC}_i \circ \texttt{MC}$$

and the last round $\mathcal{R}'_{31}$ as

$$\mathcal{R}'_{31} = \texttt{ATK}_{31} \circ \texttt{ARC}_{31} \circ \texttt{MC}.$$

We give details of the corresponding hardware implementations in Section 6.

4 Design Rationale

When designing `CRAFT`, the main criterion was to use components which are best suited for the fault-detection constructions following the structure introduced in [1], while also providing the necessary cryptographic security. The second design criterion was to build a construction for which, with least possible changes, both encryption and decryption use a similar structure. All details of the design choices are explained in the following subsections.

4.1 Involutory Building Blocks

To design a cipher with a similar structure for both encryption and decryption, we restrict our choices for the components of *substitution* and *permutation* to involutory ones. From the fact that all round operations are involutions and by applying the last linear round $\mathcal{R}'_{31}$, we made the `CRAFT` decryption a parametrized `CRAFT` encryption.

Lemma 1. *Decryption with* `CRAFT` *with tweakeys* (TK_0, TK_1, TK_2, TK_3) *and round constants* $(RC_0, \cdots, RC_{31})$ *is the same as the* `CRAFT` *encryption with tweakeys* $(TK'_3, TK'_2, TK'_1, TK'_0)$ *and round constants* $(RC_{31}, \cdots, RC_0)$*, where* $TK'_i = \texttt{MC}(TK_i)$*.*

The proof is straightforward and is based on the facts that $\texttt{PN} \circ \texttt{SB} = \texttt{SB} \circ \texttt{PN}$ and $\texttt{MC} \circ \texttt{ARC} = \texttt{ARC} \circ \texttt{MC}$. The second identity holds since the round constants are applied on the second nibble of each column while

$$\mathtt{MC}\left(\begin{bmatrix}0\\x\\0\\0\end{bmatrix}\right)=\begin{bmatrix}0\\x\\0\\0\end{bmatrix}.$$

Algorithm 4.1 and 4.2 show pseudo-code for the encryption and decryption functions respectively. Lemma 1 shows that the two algorithms can be efficiently merged. We further discuss about this feature of `CRAFT` in Section 6 with respect to hardware implementations.

Algorithm 4.1: ENCRYPTION

```
Input  : X: plaintext
         K0||K1: cipher key
         T: tweak
Output : Y: ciphertext
TK0 ← K0 ⊕ T
TK1 ← K1 ⊕ T
TK2 ← K0 ⊕ Q(T)
TK3 ← K1 ⊕ Q(T)
Y ← X
for i ← 0 to 31 do
    Y ← MC(Y)
    Y4,5 ← Y4,5 ⊕ RCi
    Y ← Y ⊕ TK(i mod 4)
    if i ≠ 31 then
        Y ← PN(Y)
        Y ← SB(Y)
    end
end
```

Algorithm 4.2: DECRYPTION

```
Input  : X: ciphertext
         K0||K1: cipher key
         T: tweak
Output : Y: plaintext
TK0 ← MC(K0 ⊕ T)
TK1 ← MC(K1 ⊕ T)
TK2 ← MC(K0 ⊕ Q(T))
TK3 ← MC(K1 ⊕ Q(T))
Y ← X
for i ← 31 to 0 do
    Y ← MC(Y)
    Y4,5 ← Y4,5 ⊕ RCi
    Y ← Y ⊕ TK(i mod 4)
    if i ≠ 0 then
        Y ← PN(Y)
        Y ← SB(Y)
    end
end
```

4.2 Sbox

To choose the Sbox which best suits our structure while at the same time having good cryptographic properties, we do the following. For all 46 206 736 involutory 4-bit Sboxes, we first evaluate their *uniformity* u and *linearity* l, i.e.,

$$u=\max_{\alpha\neq 0,\beta}|\{x\mid S(x)+S(x+\alpha)=\beta\}|,$$

$$l=2\cdot\max_{\alpha,\beta\neq 0}|\{x\mid\langle\beta,S(x)\rangle=\langle\alpha,x\rangle\}|-2^n,\quad n=4$$

and discard all Sboxes with trivial differential or linear characteristics. Second, since a bit-permutation at the input or at the output of an Sbox does not change its implementation area, we omit those Sboxes from the candidate list which are bit-permutation equivalent of each other. In other meaning, if two Sboxes are different only with respect to a bit-permutation at the input/output, we keep only one of them. Then, we evaluate the implementation area cost concerning the *independence property* introduced in [1] for the remaining Sbox candidates.

Independence Property [1] Assume a 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 coordinate 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 any two component circuits. In other words,

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

where $\mathcal{G}^i$ stands for a set of gates implementing the component-function $T^i(\cdot)$. Fulfilling the independence property guarantees the prevention of fault propagation, i.e., a faulty gate or register in such a circuit leads to at most one faulty output bit.

Therefore, the circuit implementing an Sbox needs to be split up into independent component circuits each computing exactly *one* output bit. In the implementation of the fault-detection structure – as explained in Section 2 – in addition to the Sbox S, depending on the size of the redundancy, either $F \circ S \circ F^{-1}$ or $F \circ S$ needs to be implemented (see Figure 2), with $F : x \mapsto x \cdot P$ and P the rightmost part of the generator matrix of the underlying binary linear systematic code. Considering all four cases for the redundancy size $m \in \{1, 2, 3, 4\}$, we define four redundant Sboxes $S_1 = F_1 \circ S$, $S_2 = F_2 \circ S$, $S_3 = F_3 \circ S$ and $S_4 = F_4 \circ S \circ F_4^{-1}$. The rightmost part of the corresponding generator matrices can be given as follows:

$$P_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix},\ P_2 = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix},\ P_3 = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix},\ P_4 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}.$$

Note that P_1 is that of the parity code, and P_4 the extended Hamming code. It is noteworthy that any two rows of P_4 result in a valid choice for P_2, and the same holds for P_3 made of any three rows of P_4. Therefore, in our comparisons we consider $S_4' = F_4 \circ S$, and choose the two cheapest output bits (with respect to the necessary area) as S_2 and the three cheapest bits as S_3 where by cheap we mean smaller area size in the hardware implementation.

Since we need to fulfill the independence property, the size of the implementation for a vectorial Boolean function is equal to the sum of the area for implementing each of its Boolean coordinate functions. Hence, to evaluate the size of the implementation of S, S_1, … and S_4, we need to know the size of the implementation of S, S_1, S_4' and S_4, which include 13 Boolean coordinate functions. This means that we do not need to implement and synthesize all Sbox candidates. Instead, we only need to evaluate the size of the implementation of all 12 870 four-bit Boolean coordinate functions. Actually, by omitting the Boolean coordinate functions which are bit-permutation-equivalent of the others, we end up with only 730 Boolean coordinate functions which need to be implemented and synthesized to evaluated their implementation size. To this end, using Synopsys DesignCompiler with the publicly available IBM 130 nm ASIC library we accomplished this task in a fraction of one day. Since we now have evaluated the area requirement of all 4-bit Boolean coordinate functions, we can easily calculate the implementation size of any given S, S_1, S_4' and S_4.

In order to classify the constructed Sboxes, we consider the implementation size of five combinations: S, (S, S_1), (S, S_2), (S, S_3), (S, S_4) corresponding to the Sbox itself (without redundancy), and four other cases of the redundancy size $m \in \{1, 2, 3, 4\}$. We come up with the results given in Table 3 sorted by the first being the best choice. By the class number, we refer to the index of the affine equivalent class of 4-bit Sboxes, introduced in [18], while (u, l) denote the uniformity and linearity of the corresponding class, and (n, m) the minimum number of active Sboxes in the differential and linear attack to reach a differential trail probability of $\leq 2^{-64}$ and a linear trail correlation of $\leq 2^{-32}$, respectively.

In order to obtain the list in Table 3, we searched through all cases and found no other choice for which its all five size values are smaller than that of the candidate Sbox class

Table 3: Result of the Sbox search, area size using the IBM 130 nm ASIC library.

Class	(u,l)	(n,m)	Sbox	Size [GE]				
				S	(S,S_1)	(S,S_2)	(S,S_3)	(S,S_4)
266	(4,4)	(32,32)	`9BDFAE678041C253`	11	16	22.5	26.5	26.25
262	(8,4)	(64,32)	`018945EA237DFB6C`	12.25	19.5	20.75	26	29
45	(6,6)	(46,78)	`016E4F2798ACBD35`	12	16	20	25.25	30.5
51	(6,6)	(46,78)	`B7A39F816420DCE5`	12.5	18.5	19.75	25.25	34.75
76	(6,6)	(46,78)	`E6AFD5178C2B9403`	12.75	18.75	20.5	25	34.75
208	(6,6)	(46,78)	`413205B7CEA68F9D`	12.25	19	24.25	30.25	25.75
			`D16EF52798ACB034`	13	19.25	23.75	29.75	25.75
24	(8,6)	(64,78)	`6C284E0F39BA1D57`	13	16.5	21	25.75	32.5
32	(8,6)	(64,78)	`3210654D89CBA7EF`	14	20	21	26.25	42.25
46	(8,6)	(64,78)	`B7A395816420DCEF`	12.5	17.5	19	24.25	33
57	(8,6)	(64,78)	`DF574263B9A8E0C1`	11.25	15.25	22	28	28.25
101	(8,6)	(64,78)	`21034D6CAF8E75B9`	14	17.75	21.25	25.5	32.25
102	(8,6)	(64,78)	`290D6B4C81A573FE`	13	19.75	25	31.5	25
			`DF32CE6A987B4051`	12	19.25	20.25	26.5	25
137	(10,6)	(95,78)	`01CD6F4E89AB2375`	19.25	21.25	21.25	25.75	32
			`98DCF65710BA32E4`	19.5	20	21.25	25.75	33.25
141	(10,6)	(95,78)	`98DCF76510BA32E4`	12.75	19	21.5	26	32.25
149	(10,6)	(95,78)	`465E021FCDAB8937`	12	17.75	20.5	26	26.75
217	(10,6)	(95,78)	`6523410BEDF7C98A`	11	16.25	20.25	22.25	31.25
			`6DF74903E5ABC182`	14	21.5	20.25	20.5	35.75
216	(12,6)	(155,78)	`2301ABF7CD4589E6`	10.75	15.5	21	27.5	25.75
			`A68437152B09FDEC`	11	15.25	18.5	23.5	29.75
			`E6C43715B9A82F0D`	11.25	16.5	20.75	26.25	27.75
			`6789AB012345FDEC`	11.75	16.5	19.25	23.5	27
			`84A617350B29FDEC`	12.5	15.75	20	25.25	29.25
			`210BFD8967A3E5C4`	12.5	20.75	21	27.25	25.5
			`EAC8654F3B192D07`	13	18.75	23	29	25.5

266, the so-called reference Sbox. In the list we included the other cases which have at least one size (amongst five) smaller than the reference Sbox.

Notably, the reference Sbox has the minimum uniformity and linearity. Although other candidates also lead to low area requirements, they have larger uniformity or linearity. Hence, their usage would imply that we should use more rounds to protect the cipher from differential or linear attacks. Therefore, we decided to choose the reference Sbox as the best suited choice to our structure for fault detection. The reference Sbox is a bit-permuted version of the `MIDORI` Sbox. It is a coincidence since the `MIDORI` Sbox has been selected based on its predicted low energy consumption. Hence, to avoid introducing a new Sbox we just use the `MIDORI` Sbox in `CRAFT`.

4.3 Linear Layer

For making `CRAFT` efficient to be implemented in fault-detection structures with redundancy size $m < 4$, we decided to use a binary matrix for the `MixColumn` operation. According to [1], it allows the `MixColumn` of the redundant part of the circuit A' to solely operate on the redundant part of the information (see [1, Theorem 1] and Figure 2(b)). Among all 20 160 bijective 4×4 binary matrices, only 316 are involutions. On the other hand, since `CRAFT` includes `PermuteNibbles` applied right after `MC`, and because we check all the involutory permutations for each matrix, we reduce this set of 316 matrices to 30 candidates which are equivalent up to a permutation of the rows.

Lemma 2. *Let $\mathcal{P}_r$ be a permutation over the rows of the state. The encryption with round operations* SB*,* MC *and* PN *with matrix M and permutation $\mathcal{P}$ is the same as the encryption with round operations* SB*,* MC′ *and* PN′ *with modified tweakey and round constants and up to a nibble-wise permutation of the plaintext and ciphertext, where* MC′ *applies $M' = \mathcal{P}_r^{-1} \circ M \circ \mathcal{P}_r$ and* PN′ *applies $\mathcal{P}' = \mathcal{P}_r^{-1} \circ \mathcal{P} \circ \mathcal{P}_r$.*

The 30 candidates for the matrix M are given below.

$$M_0 : \begin{bmatrix} \texttt{1000} \\ \texttt{0100} \\ \texttt{0010} \\ \texttt{0001} \end{bmatrix} \quad M_1 : \begin{bmatrix} \texttt{0100} \\ \texttt{1000} \\ \texttt{0010} \\ \texttt{0001} \end{bmatrix} \quad M_2 : \begin{bmatrix} \texttt{0100} \\ \texttt{1000} \\ \texttt{0001} \\ \texttt{0010} \end{bmatrix} \quad M_3 : \begin{bmatrix} \texttt{1001} \\ \texttt{0100} \\ \texttt{0010} \\ \texttt{0001} \end{bmatrix} \quad M_4 : \begin{bmatrix} \texttt{0100} \\ \texttt{1000} \\ \texttt{0011} \\ \texttt{0001} \end{bmatrix}$$

$$M_5 : \begin{bmatrix} \texttt{1011} \\ \texttt{0100} \\ \texttt{0010} \\ \texttt{0001} \end{bmatrix} \quad M_6 : \begin{bmatrix} \texttt{1001} \\ \texttt{0101} \\ \texttt{0010} \\ \texttt{0001} \end{bmatrix} \quad M_7 : \begin{bmatrix} \texttt{1010} \\ \texttt{0101} \\ \texttt{0010} \\ \texttt{0001} \end{bmatrix} \quad M_8 : \begin{bmatrix} \texttt{0101} \\ \texttt{1001} \\ \texttt{0010} \\ \texttt{0001} \end{bmatrix} \quad M_9 : \begin{bmatrix} \texttt{1110} \\ \texttt{0010} \\ \texttt{0100} \\ \texttt{0001} \end{bmatrix}$$

$$M_{10} : \begin{bmatrix} \texttt{1111} \\ \texttt{0100} \\ \texttt{0010} \\ \texttt{0001} \end{bmatrix} \quad M_{11} : \begin{bmatrix} \texttt{1001} \\ \texttt{0101} \\ \texttt{0011} \\ \texttt{0001} \end{bmatrix} \quad M_{12} : \begin{bmatrix} \texttt{0110} \\ \texttt{1001} \\ \texttt{0001} \\ \texttt{0010} \end{bmatrix} \quad M_{13} : \begin{bmatrix} \texttt{1011} \\ \texttt{0101} \\ \texttt{0010} \\ \texttt{0001} \end{bmatrix} \quad M_{14} : \begin{bmatrix} \texttt{0101} \\ \texttt{1001} \\ \texttt{0011} \\ \texttt{0001} \end{bmatrix}$$

$$M_{15} : \begin{bmatrix} \texttt{1111} \\ \texttt{0010} \\ \texttt{0100} \\ \texttt{0001} \end{bmatrix} \quad M_{16} : \begin{bmatrix} \texttt{1011} \\ \texttt{0111} \\ \texttt{0010} \\ \texttt{0001} \end{bmatrix} \quad M_{17} : \begin{bmatrix} \texttt{0111} \\ \texttt{1011} \\ \texttt{0010} \\ \texttt{0001} \end{bmatrix} \quad M_{18} : \begin{bmatrix} \texttt{1110} \\ \texttt{0101} \\ \texttt{0011} \\ \texttt{0001} \end{bmatrix} \quad M_{19} : \begin{bmatrix} \texttt{0111} \\ \texttt{1010} \\ \texttt{0011} \\ \texttt{0001} \end{bmatrix}$$

$$M_{20} : \begin{bmatrix} \texttt{1110} \\ \texttt{0011} \\ \texttt{0101} \\ \texttt{0001} \end{bmatrix} \quad M_{21} : \begin{bmatrix} \texttt{1011} \\ \texttt{0111} \\ \texttt{0001} \\ \texttt{0010} \end{bmatrix} \quad M_{22} : \begin{bmatrix} \texttt{1101} \\ \texttt{0111} \\ \texttt{0001} \\ \texttt{0010} \end{bmatrix} \quad M_{23} : \begin{bmatrix} \texttt{0111} \\ \texttt{1011} \\ \texttt{0001} \\ \texttt{0010} \end{bmatrix} \quad M_{24} : \begin{bmatrix} \texttt{1111} \\ \texttt{0101} \\ \texttt{0011} \\ \texttt{0001} \end{bmatrix}$$

$$M_{25} : \begin{bmatrix} \texttt{1111} \\ \texttt{0011} \\ \texttt{0101} \\ \texttt{0001} \end{bmatrix} \quad M_{26} : \begin{bmatrix} \texttt{1111} \\ \texttt{1010} \\ \texttt{1011} \\ \texttt{0110} \end{bmatrix} \quad M_{27} : \begin{bmatrix} \texttt{1110} \\ \texttt{1101} \\ \texttt{1011} \\ \texttt{0111} \end{bmatrix} \quad M_{28} : \begin{bmatrix} \texttt{1110} \\ \texttt{1011} \\ \texttt{1101} \\ \texttt{0111} \end{bmatrix} \quad M_{29} : \begin{bmatrix} \texttt{0111} \\ \texttt{1011} \\ \texttt{1101} \\ \texttt{1110} \end{bmatrix}$$

For each of these 30 candidate matrices, in principle, we need to search through all 46 206 736 involutory permutations for PN. The lemma given below explains that it is not necessary to consider *all* such permutations, but only those up to a permutation over the columns of the state.

Lemma 3. *Let $\mathcal{P}_c$ be a permutation over the columns of the state. The encryption with round operations* SB*,* MC *and* PN *with permutation $\mathcal{P}$ is the same as the encryption with round operations* SB*,* MC *and* PN′ *with modified tweakey and round constants and up to a nibble-wise permutation of the plaintext and ciphertext, where* PN′ *applies $\mathcal{P}' = \mathcal{P}_c^{-1} \circ \mathcal{P} \circ \mathcal{P}_c$.*

Therefore, we can reduce the search space of all involutory permutations $\mathcal{P}$ by the above equivalency. In combination with 30 candidate matrices M_0 to M_{29}, we need to search through such permutations in order to find those which provide the highest possible security level. To this end, we evaluated the minimum number of rounds such that the linear layer attains full diffusion in both forward and backward directions (r_1) together with the minimum number of rounds to guarantee at least 32 active Sboxes in both differential and linear attacks (r_2).

Regardless of the choice for involutory $\mathcal{P}$, the matrices $M_0, \dots, M_{11}$ do not provide full diffusion. For the remaining 18 matrices, we list their minimum possible r_1 and r_2 together with their hardware implementation cost in Table 4. By the implementation cost, we refer to: the number of 2-input XOR gates (*#xor*), and the number of the 2-to-1 multiplexers (*#mux*) needed to implement the tweakey schedule of both encryption and decryption together. Note that for encryption, the tweakey TK_i is added to the state

Table 4: Implementation cost and security properties of the candidate matrices for `MC`

M_i	12	13	14/15	16/17	18	19/20	21	22	23	24	25	26	27/28/29
$\#xor$	2	3	3	4	4		4			5		7	8
$\#mux$	4	2	3	2	3		4			3		4	4
r_1	8	7	7	5	6	6	5	6	5	4	4	4	4
r_2	10	9	11	16	9	8	16	8	8	8	11	8	7

while in decryption $\texttt{MC}(TK_i)$ is, hence $\#mux$ is only considered when the implementation should support both encryption and decryption. It is noteworthy to emphasize that the numbers given for $\#xor$ and $\#mux$ are those required for one column of bits. Hence for one complete block, we need to multiply these values by 16.

From the remaining matrices, M_{12} is the smallest one with respect to its implementation cost in the encryption-only case, while M_{13} is the smallest if both encryption and decryption should be supported. Since M_{13} can provide better security properties, we choose it as the matrix of `MC` of `CRAFT`.

Over all involutory permutations, only the following candidate for $\mathcal{P}$ provides the reported r_1 and r_2. More precisely, it attains full diffusion after 7 rounds and assures at least 32 active Sboxes after 9 rounds in both differential and linear attacks.

$$\mathcal{P} = [15, 12, 13, 14,\ \ 10, 9, 8, 11,\ \ 6, 5, 4, 7,\ \ 1, 2, 3, 0]$$

This permutation replaces the nibbles in the first row with the nibbles in the last row and also the nibbles in the second row with those in the third row. Then it does a right (resp. left) shift in the first (resp. fourth) row and a shuffle in the second and third rows (see Figure 4).

4.4 Round Constants

We decided to use LFSRs to generate the round constants, since compared to a randomly chosen set of round constants, an LFSR usually leads to lower implementation cost. Further, to make it efficient with respect to the considered fault-detection mechanism, we restrict the LFSR size to 4 bit as $k = 4$ in the underlying code $[l, k, d]$ (due to the Sbox size). The LFSR can also be considered as the round counter. This implies that the period of the LFSR should be larger than the number of rounds. As an n-bit LFSR has maximum period of $2^n - 1$, one 4-bit LFSR does not suffice. Hence, we decided to use two LFSRs, one with a 4-bit and one with a 3-bit state. By using primitive polynomials for their feedback functions, the joint period of the LFSRs can reach $15 \cdot 7 = 105$ which is more than enough. While there is only one primitive polynomial for a 3-bit LFSR ($x^3 + x + 1$), there are two choices for the 4-bit one ($x^4 + x + 1$ and $x^4 + x^3 + 1$). With respect to the size of their hardware implementation, there is no preference between the 4-bit polynomials. We have just chosen $x^4 + x + 1$ as the polynomial for the 4-bit LFSR.

In every round, we add the states of 4-bit and 3-bit LFSRs to the fourth and fifth nibbles of the state of the cipher. We choose these two positions, since – considering our chosen linear layer – nibbles in the first/second row of the state have full diffusion after 5/6 rounds. Actually, they get involved in the entire state nibbles after 5/6 rounds while this happens after 7 rounds if the round constants are added to the third row. Although, the first row has the fastest diffusion, adding round constants to this row causes larger latency in each clock cycle compared to adding them to the second row. Hence, we decide to add them in the fourth and fifth nibble of the cipher.

4.5 Key and Tweak Schedule

To make the key schedule of the cipher small and lightweight, we decided to use the same round keys in an alternating way. In particular, by separating the 128-bit master key into two 64-bit halves K_0 and K_1, using 64 multiplexers we use K_0 in the even rounds and K_1 in the odd rounds. This is beneficial in a round-based implementation (as it is our target design architecture) since no extra register is required to process and generate the round keys. The same technique has been used in PICCOLO and MIDORI. As a side note, in order to make sure that the first and the last round keys are different, the total number of rounds has to be even.

To make use of the same concept for the tweak schedule, we decided to not use an arbitrary update function for the tweak. Instead, using a permutation Q on the nibbles of the tweak T we calculate $Q(T)$ and iteratively add it to the round key. Actually, we use T in the first two rounds and $Q(T)$ for the next two rounds and iterate this order. In a round-based implementation this needs only 64 XOR and 64 MUX extra logic.

To find a permutation Q which provides the highest security, we evaluated the necessary number of rounds to assure that there are 32 active Sboxes with regard to the related-tweak differential attack [49]. On the other hand, to make Time-Data-Memory trade-off attacks less powerful, we decided to use a circular permutation. The reason for this restriction is explained in Section 5.3 in more detail.

Since the search space for a circular permutation is $15! \approx 2^{40}$, it is not possible to check all permutations. Hence, we decided to choose about one thousand randomly generated permutations and examine their security. We found that the permutation given in Section 3 that guarantees 32 active Sboxes in 13 rounds, which is less than the one for other permutations.

5 Security Analysis of CRAFT

In this section, we provide a detailed analysis of the security of CRAFT. In fact, since the general structure of CRAFT is similar to that of AES, MIDORI, SKINNY and MANTIS, the security analysis of CRAFT is also more or less similar to that of those primitives.

5.1 Security Claim

Overall, we claim 124 bit security of CRAFT in the related-tweak model.

From the provided analyses below, the most promising cryptanalysis on CRAFT is a an accelerated exhaustive search with a time complexity of 2^{124} cipher encryptions, and data and memory complexity of 16 (see Section 5.2).

We expect that a 30-round version of CRAFT attains 128 bit security against (impossible) differential and (zero-correlation) linear attacks, integral attacks and invariant attacks as well as Meet-in-the-Middle attack. It is noteworthy that it does not mean that there actually exists a 30-round attack on CRAFT. It is only an upper bound on the number of rounds that can be attacked.

Hence, considering two extra rounds as a security margin, we claim that 32-round CRAFT has 124-bit security in the related-tweak model. We do not claim any security in the chosen-key, known-key or related-key model.

5.2 Exhaustive Search

Due to the simple tweakey schedule of CRAFT, there are some deterministic related-key related-tweak characteristics which can accelerate the typical exhaustive search attack. Consider K_0, K_1 and T as two halves of the key and tweak, respectively and $K_0' = K_0 + \Delta$,

$K'_1 = K_1 + \Delta$ and $T' = T + \Delta$ where $\Delta = (x, x, \dots, x)$ and $x \in \mathbb{F}_2^4$. Then we have $Q(\Delta) = \Delta$ which cause the relations below.

$$
\begin{aligned}
TK'_0 &= K'_0 + T' = (K_0 + \Delta) + (T + \Delta) = K_0 + T = TK_0,\\
TK'_1 &= K'_1 + T' = (K_1 + \Delta) + (T + \Delta) = K_1 + T = TK_1,\\
TK'_2 &= K'_0 + Q(T') = (K_0 + \Delta) + (Q(T) + \Delta) = K_0 + Q(T) = TK_2,\\
TK'_3 &= K'_1 + Q(T') = (K_1 + \Delta) + (Q(T) + \Delta) = K_1 + Q(T) = TK_3.
\end{aligned}
$$

This means encryption under two different tweakey tuples of (K_0, K_1, T) and (K'_0, K'_1, T') are the same. Using these deterministic characteristics, the attacker can accelerate the exhaustive search by a factor of 2^4. First, he asks for encryption of the same plaintext P under 16 different tweaks of $T, T+\Delta_1, \dots, T+\Delta_{15}$ where $\Delta_x = (x, x, \dots, x)$ that we show the corresponding ciphertexts by $C_0, C_1, \dots, C_{15}$. Then, by setting one of the key nibbles to constant value of zero, for each of 2^{124} possible key candidate (K_0^*, K_1^*), he computes C^*, the encryption of P using K_0^*, K_1^* and T. If C^* is equal to C_x, then $(K_0^* + \Delta_x,\, K_1^* + \Delta_x)$ is a candidate for the master key. This way, there will remain only about 2^{64} key candidates for the master key which a check on another pair of plaintext and ciphertext will determine the only candidate for the master key. All together, exhaustive search attack on `CRAFT` using 16 chosen plaintext data has complexity of 2^{124} encryptions.

5.3 Time-Data-Memory Trade-off Attacks

Since `CRAFT` uses a simple tweakey schedule, it is necessary to analyze the security of the cipher against *Time-Data-Memory Trade-off (TDM TO) attacks* [32]. Actually, some choices for the permutation Q in the tweak schedule give an opportunity to the attacker to do a TDM TO attack. In the following we describe a TDM TO attack that helps us to choose the permutation Q carefully.

In the offline phase, the attacker fixes the tweakeys to $TK_0 = 0, TK_1 = X, TK_2 = T'$ and $TK_3 = X + T'$ which means:

$$K_0 = T \ , \ K_0 + K_1 = X \ , \ T + Q(T) = T'.$$

For a fixed plaintext P_0 and all possible values of X and T', he computes the corresponding ciphertext $C_{T',X}$ and saves the X value in the index (T', C) of a table $\mathcal{T}$. In the online phase, by asking the encryption for a plaintext P_0 and for all possible tweaks T, he receives the corresponding ciphertext C_T. Then for each value of T, he gets a candidate for $K_0 + K_1$ by looking up to the index $(T + Q(T), C_T)$ of $\mathcal{T}$. By doing an exhaustive search on the 2^{64} key candidates, he can find the correct value of the 128-bit key. This attack requires $2^{64+\dim\{T+Q(T)\}}$ pre-computations, $2^{64+\dim\{T+Q(T)\}}$ memory, 2^{65} online computions and 2^{64} data. But with some small modifications, it can be changed to all online attack with $2^{64+\dim\{T+Q(T)\}}$ computations, 2^{64} data and memory.

Since a circular permutation Q has the maximum value for $\dim\{T + Q(T)\} = 60$ we decided to use such a Q that improves the security of `CRAFT` against this attack. In other words, the TDM TO attack on `CRAFT` has 2^{124} time, 2^{64} data and memory complexity.

5.4 Differential and Linear Cryptanalysis

In order to argue for the resistance of `CRAFT` against *differential* and *linear attacks*, we computed lower bounds on the minimum number of active Sboxes, both in the single-tweak and related-tweak model. We recall that, in a differential (resp. linear) characteristic, an Sbox is called *active* if the input difference (resp. mask) is non-zero. In contrast to the single-tweak model, where the tweak is constant and thus does not change the activity pattern, an attacker is allowed to introduce differences or masks within the tweak state in the related-tweak model. It is noteworthy that such an attacker model is considered as the most important model when examining the security of a tweakable block cipher. Note that we don't claim any security in the related-key model.

Table 5 shows the lower bounds on the minimum number of active Sboxes in the single-tweak model (**ST**) and the related-tweak model (**RT**) for 1 up to 17 rounds. In order to compute these bounds, we used the two techniques of Matsui's recursive algorithm as explained in [65] and Mixed-Integer Linear Programming as explained in [67, 83]. It is noteworthy that both of the approaches take only the properties of the linear layer into account and are independent of the specification of the Sbox. After 13 rounds, all bounds are high enough in order to ensure that no distinguisher based on a single (related-tweak) differential (respectively linear) characteristic exists. Since the maximum differential probability (resp. absolute linear correlation) for an active Sbox is 2^{-2} (resp. 2^{-1}), having at least 32 active Sboxes will cause the probability (resp. absolute correlation) of a differential (resp. linear) characteristic to be less than or equal to 2^{-64} (resp. 2^{-32}). Hence, such a characteristic is not useful to distinguish the cipher from a random permutation. Since there is no cancellation of active Sboxes in linear characteristics in the related-tweak model, we only considered the single-tweak model for the linear attack [59]. Thus, the bounds for **ST** give valid bounds also for the **RT** case.

Table 5: Lower bounds on the minimum number of active Sboxes up to 17 rounds. $\mathbf{RT}_i$ refers to the characteristic starting with round $\mathcal{R}_{4j+i}$.

Model	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Linear	1	2	4	6	10	14	20	26	32	36	40	44	48	52	56	60	64
ST Diff.	1	2	4	6	10	14	20	26	32	36	40	44	48	52	56	60	64
$\mathbf{RT}_0$ **Diff.**	0	1	2	4	6	12	14	19	22	25	27	32	36	38	40	46	49
$\mathbf{RT}_1$ **Diff.**	0	1	2	5	7	10	15	18	22	24	28	32	35	38	43	45	44
$\mathbf{RT}_2$ **Diff.**	0	1	2	4	6	12	16	19	21	24	27	30	34	39	41	42	44
$\mathbf{RT}_3$ **Diff.**	0	1	2	5	7	10	15	18	21	24	28	31	34	38	39	41	47

Note that only a single characteristic is considered in analysis so far. Thus, the distinguishers might actually be stronger due to differential (resp. linear hull) effects. To have a better estimation about the probability of differentials (resp. average square correlation of linear hulls), by fixing input and output differences (resp. masks) in the differential (resp. linear hull), we found all different single characteristics which follow the same Sbox activity pattern with the minimum number of active Sboxes. Then by summing all the probabilities (resp. square correlations) of each single characteristics, we found a lower bound for the probability of corresponding differential (resp. average square correlation of linear hull[8]). Note that it is a lower bound, because for a fixed input and output difference (resp. mask), there might be some other single characteristics that are not following the Sbox activity pattern. However, as for such characteristics the number of active Sboxes will be higher, we assume their affect on the probability of differential (resp. average square correlation of linear hull) to be negligible.

In the **ST** case, for 9-round `CRAFT` which has at least 32 active Sboxes, we found four optimal differentials, each having a probability of $2^{-54.67}$. Similarly, we found four optimal linear hulls, each having an average square correlation of $2^{-40.95}$. Using the same method, we computed the highest probability and the average square correlation for a higher number of rounds. In the **ST** differential case, for 10 round `CRAFT`, we found the

[8] In our considerations we assume that all linear trails contribute to the linear hull with the same sign, i.e., each trail can only increase the average square correlation. Note that this is a worst-case consideration as we can expect that the correlations will appear with different signs.

following eight differentials with probability of $2^{-62.61}$:

$$(a,0,a,a,0,0,a,a,0,0,0,0,0,0,a,a) \rightarrow (a,0,0,0,0,0,0,0,0,0,0,0,0,0,a,a)$$
$$(0,0,a,0,a,0,a,0,0,0,0,0,a,0,a,0) \rightarrow (a,0,0,0,0,0,0,0,0,0,0,0,a,0,a,0)$$
$$(0,a,a,a,0,0,a,a,0,0,0,0,0,0,a,a) \rightarrow (0,a,0,0,0,0,0,0,0,0,0,0,0,0,a,a)$$
$$(0,0,0,a,0,a,0,a,0,0,0,0,0,a,0,a) \rightarrow (0,a,0,0,0,0,0,0,0,0,0,0,0,a,0,a)$$
$$(a,a,a,0,a,a,0,0,0,0,0,0,a,a,0,0) \rightarrow (0,0,a,0,0,0,0,0,0,0,0,0,a,a,0,0)$$
$$(a,0,0,0,a,0,a,0,0,0,0,0,a,0,a,0) \rightarrow (0,0,a,0,0,0,0,0,0,0,0,0,a,0,a,0)$$
$$(a,a,0,a,a,a,0,0,0,0,0,0,a,a,0,0) \rightarrow (0,0,0,a,0,0,0,0,0,0,0,0,a,a,0,0)$$
$$(0,a,0,0,0,a,0,a,0,0,0,0,0,a,0,a) \rightarrow (0,0,0,a,0,0,0,0,0,0,0,0,0,a,0,a)$$

In the **ST/RT** linear case, for 14 round `CRAFT`, we found the following eight linear hulls with average square correlation of $2^{-62.12}$:

$$(0,5,5,0,0,0,0,0,0,5,5,0,0,5,5,5) \rightarrow (0,5,5,0,0,0,0,0,0,0,0,0,0,0,0,5)$$
$$(0,5,0,5,0,0,0,0,0,5,0,5,0,5,0,0) \rightarrow (0,5,0,5,0,0,0,0,0,0,0,0,0,0,0,5)$$
$$(5,0,0,5,0,0,0,0,5,0,0,5,5,0,5,5) \rightarrow (5,0,0,5,0,0,0,0,0,0,0,0,0,0,5,0)$$
$$(5,0,5,0,0,0,0,0,5,0,5,0,5,0,0,0) \rightarrow (5,0,5,0,0,0,0,0,0,0,0,0,0,0,5,0)$$
$$(0,5,0,5,0,0,0,0,0,5,0,5,0,0,0,5) \rightarrow (0,5,0,5,0,0,0,0,0,0,0,0,0,5,0,0)$$
$$(5,0,0,5,0,0,0,0,5,0,0,5,5,5,0,5) \rightarrow (5,0,0,5,0,0,0,0,0,0,0,0,0,5,0,0)$$
$$(0,5,5,0,0,0,0,0,0,5,5,0,5,5,5,0) \rightarrow (0,5,5,0,0,0,0,0,0,0,0,0,5,0,0,0)$$
$$(5,0,5,0,0,0,0,0,5,0,5,0,0,0,5,0) \rightarrow (5,0,5,0,0,0,0,0,0,0,0,0,5,0,0,0)$$

In order to recover the key, several rounds can be appended before and after the differentials or linear hulls. The number of appended rounds depends on the minimum number of rounds that achieve full diffusion, i.e., 7 rounds for `CRAFT`. Hence, the attacker can add at most 7 rounds both at the beginning and at the end of the characteristic. Thus, the total number of appended rounds can be at most $2 \times 7 = 14$ rounds. Therefore, we expect that an attacker cannot have a successful single-tweak differential attack on more than $10 + 14 = 24$ rounds and (single-/related-tweak) linear attack on more than $14 + 14 = 28$ rounds. It is noteworthy that 14 rounds for appending to the trails is an upper bound and is not always possible. For example, for the above mentioned differentials and linear hulls, the maximum possible number of rounds to append is 7. It means that using those differentials or linear hulls the attacker cannot have a successful single-tweak differential attack on 18 rounds or a (single-/related-tweak) linear attack on 22 rounds.

In the related-tweak cases, the differentials are dependent on the starting round, i.e., the index of RT. For each $0 \leq i \leq 3$, in a process similar to the single-tweak case, we found the below differentials as the longest ones with a probability higher than 2^{-64}:

RT$_0$: 15-round with a probability of $2^{-55.14}$:

$$(0,0,0,0,a,0,0,0,0,0,0,0,0,0,0,0) \rightarrow (0,0,0,0,0,0,0,0,0,0,a,0,a,0,0,0)$$

RT$_1$: 16-round with a probability of $2^{-57.18}$:

$$(0,a,0,a,0,0,a,a,0,0,0,0,0,0,0,a) \rightarrow (0,0,0,0,0,0,0,a,a,0,0,0,0,0,0,0)$$

RT$_2$: 17-round with a probability of $2^{-60.14}$:

$$(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \rightarrow (0,0,0,0,0,0,0,0,0,0,a,0,a,0,0,0)$$

RT$_3$: 16-round with probability of $2^{-55.14}$:

$$(0,0,0,0,0,0,0,0,0,0,0,0,a,a,0,0) \rightarrow (0,0,0,0,0,0,0,0,0,0,a,0,a,0,0,0)$$

where for all of them ΔT is equal to $(0,0,0,0,0,0,0,0,0,0,a,0,0,0,0,0)$.

For the key recovery the number of rounds that can be appended for an RT_i differential is at most $4+i$ rounds before and 7 rounds after the differential. But for the above differentials, this number can be less than or equal to 10, 10, 12 and 13 rounds, respectively. Therefore, we expect that an attacker cannot have a successful related-tweak differential attack on 30 rounds.

It is noteworthy that in our entire analyses we did not consider the time complexity to check the feasibility of the attacks. Hence, the number of rounds that can be analyzed by the attacker is an upper bound.

5.5 Impossible Differentials and Zero-Correlation Linear Hulls

The structure of `CRAFT` is quite similar to `SKINNY`. Both ciphers employ very sparse and binary `MixColumn` operations and there exist several activity patterns that deterministically propagate through it. Because of that, there might exist distinguishers based on impossible differentials [54, 15] over a high number of rounds. A pair of differences (Γ, Δ) is said to be an *impossible differential* over an encryption function F if, for all plaintexts x, $F(x) + F(x+\Gamma) \neq \Delta$. Such a distinguisher over a reduced-round version of the cipher might be used for a key-recovery attack over a larger number of rounds by filtering all the key candidates which lead to the intermediate state values with differences Γ and Δ, i.e., the intermediate state values fulfilling the impossible differential. Indeed, for `SKINNY`, among the conducted cryptanalytic attacks so far the bests are based on impossible differentials (see [12, Section 4.3] and [5, 62]). Therefore, it is crucial to evaluate the resistance of `CRAFT` against those attacks.

With the Mixed-Integer Linear Programming approach (see [27, 76]), we searched for (truncated)[9] impossible differentials over reduced-round versions of `CRAFT`. Thereby, we constrained both the input and output activity patterns to have at most two active nibbles, respectively. The largest number of rounds for which we found an impossible differential was 13. In total, we found the following twelve different 13-round truncated impossible differentials, where γ and δ can take any non-zero difference in $\mathbb{F}_2^4$. The first one is depicted in more detail in Figure 5 where a black cell indicates an active nibble (i.e., a non-zero difference), white indicates a passive nibble (i.e., a zero difference), and gray indicates a nibble that might be active or passive.

$$
\begin{aligned}
(0,0,0,\gamma,0,0,0,0,0,0,0,\gamma,0,0,0,0) &\to (0,0,0,0,0,0,0,0,\delta,0,0,0,0,0,0,0)\\
(0,0,0,\gamma,0,0,0,0,0,0,0,\gamma,0,0,0,0) &\to (0,0,0,0,0,0,0,0,0,\delta,0,0,0,0,0,0)\\
(0,0,0,\gamma,0,0,0,0,0,0,0,\gamma,0,0,0,0) &\to (0,0,0,0,0,0,0,0,0,0,\delta,0,0,0,0,0)\\
(0,0,\gamma,0,0,0,0,0,0,0,\gamma,0,0,0,0,0) &\to (0,0,0,0,0,0,0,0,0,0,0,\delta,0,0,0,0)\\
(0,0,\gamma,0,0,0,0,0,0,0,\gamma,0,0,0,0,0) &\to (0,0,0,0,0,0,0,0,\delta,0,0,0,0,0,0,0)\\
(0,0,\gamma,0,0,0,0,0,0,0,\gamma,0,0,0,0,0) &\to (0,0,0,0,0,0,0,0,0,\delta,0,0,0,0,0,0)\\
(0,\gamma,0,0,0,0,0,0,0,\gamma,0,0,0,0,0,0) &\to (0,0,0,0,0,0,0,0,0,0,0,\delta,0,0,0,0)\\
(0,\gamma,0,0,0,0,0,0,0,\gamma,0,0,0,0,0,0) &\to (0,0,0,0,0,0,0,0,\delta,0,0,0,0,0,0,0)\\
(0,\gamma,0,0,0,0,0,0,0,\gamma,0,0,0,0,0,0) &\to (0,0,0,0,0,0,0,0,0,0,\delta,0,0,0,0,0)\\
(\gamma,0,0,0,0,0,0,0,\gamma,0,0,0,0,0,0,0) &\to (0,0,0,0,0,0,0,0,0,0,0,\delta,0,0,0,0)\\
(\gamma,0,0,0,0,0,0,0,\gamma,0,0,0,0,0,0,0) &\to (0,0,0,0,0,0,0,0,0,\delta,0,0,0,0,0,0)\\
(\gamma,0,0,0,0,0,0,0,\gamma,0,0,0,0,0,0,0) &\to (0,0,0,0,0,0,0,0,0,0,\delta,0,0,0,0,0)\ .
\end{aligned}
$$

Using the same approach as for impossible differentials (by considering the transpose of M instead of M), we were looking for *zero-correlation linear hulls* [20] over reduced-round

[9] Note that our approach only takes the properties of the linear layer into account and is independent of the choice of the Sbox.

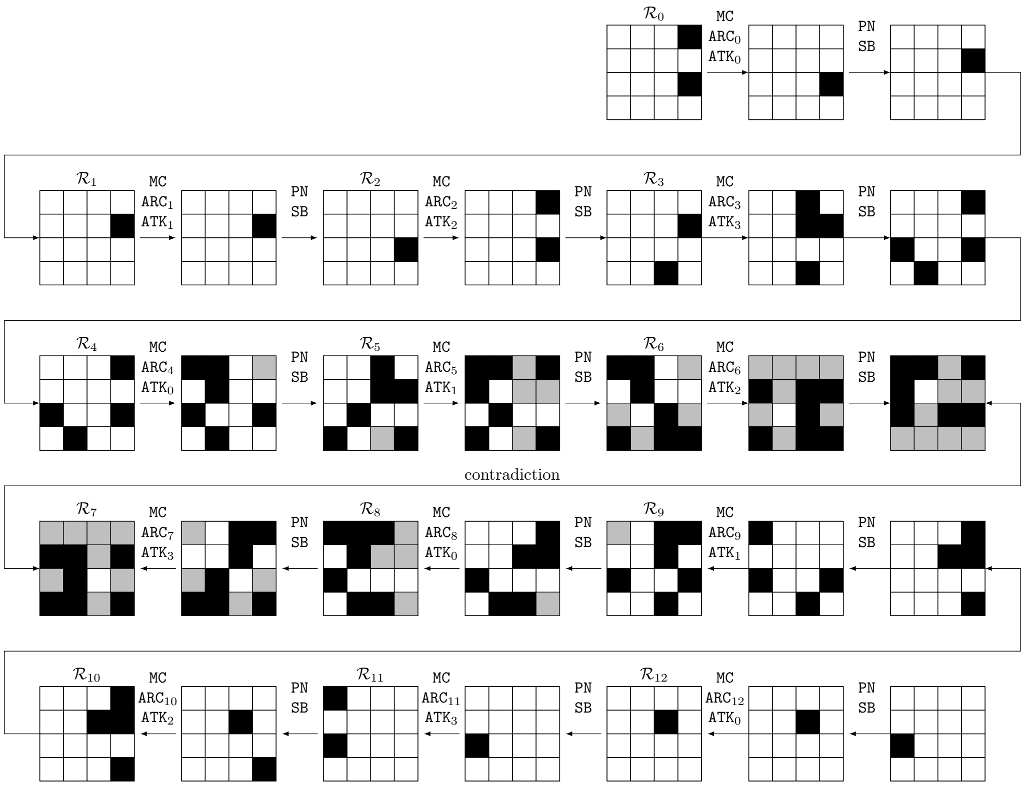


Figure 5: A 13-round impossible differential characteristic for `CRAFT`. Note that active nibbles of the plaintext difference have the same value.

versions of `CRAFT`. Again, the longest distinguisher that we found covers 13 rounds and we found the following twelve (truncated) zero-correlation linear hulls:

$$
\begin{aligned}
&(0,0,0,0,0,0,0,\gamma,0,0,0,0,0,0,0,\gamma) \to (0,0,0,0,0,0,\delta,0,0,0,0,0,0,0,0,0)\\
&(0,0,0,0,0,0,0,\gamma,0,0,0,0,0,0,0,\gamma) \to (0,0,0,0,0,\delta,0,0,0,0,0,0,0,0,0,0)\\
&(0,0,0,0,0,0,0,\gamma,0,0,0,0,0,0,0,\gamma) \to (0,0,0,0,\delta,0,0,0,0,0,0,0,0,0,0,0)\\
&(0,0,0,0,0,0,\gamma,0,0,0,0,0,0,0,\gamma,0) \to (0,0,0,0,0,\delta,0,0,0,0,0,0,0,0,0,0)\\
&(0,0,0,0,0,0,\gamma,0,0,0,0,0,0,0,\gamma,0) \to (0,0,0,0,\delta,0,0,0,0,0,0,0,0,0,0,0)\\
&(0,0,0,0,0,0,\gamma,0,0,0,0,0,0,0,\gamma,0) \to (0,0,0,0,0,0,0,\delta,0,0,0,0,0,0,0,0)\\
&(0,0,0,0,0,\gamma,0,0,0,0,0,0,0,\gamma,0,0) \to (0,0,0,0,\delta,0,0,0,0,0,0,0,0,0,0,0)\\
&(0,0,0,0,0,\gamma,0,0,0,0,0,0,0,\gamma,0,0) \to (0,0,0,0,0,0,0,\delta,0,0,0,0,0,0,0,0)\\
&(0,0,0,0,0,\gamma,0,0,0,0,0,0,0,\gamma,0,0) \to (0,0,0,0,0,0,\delta,0,0,0,0,0,0,0,0,0)\\
&(0,0,0,0,\gamma,0,0,0,0,0,0,0,\gamma,0,0,0) \to (0,0,0,0,0,0,0,\delta,0,0,0,0,0,0,0,0)\\
&(0,0,0,0,\gamma,0,0,0,0,0,0,0,\gamma,0,0,0) \to (0,0,0,0,0,0,\delta,0,0,0,0,0,0,0,0,0)\\
&(0,0,0,0,\gamma,0,0,0,0,0,0,0,\gamma,0,0,0) \to (0,0,0,0,0,\delta,0,0,0,0,0,0,0,0,0,0),
\end{aligned}
$$

where γ and δ can take any non-zero mask in $\mathbb{F}_2^4$.

By appending at most 14 rounds before and after the impossible differentials or zero-correlation linear hulls (reason for 14 rounds is explained at the end of Section 5.4), the attacker can have successful key-recovery on a higher number of rounds. Actually, for the reported characteristics it is possible to only append 12 rounds. Thus, we conclude that the attacker may have a successful attack on *at most* $13 + 13 = 26$ rounds.

5.6 Meet-in-the-Middle Attacks

To discuss the resistance of CRAFT against *Meet-in-the-middle attacks* [32], we use a similar approach as in its application to the SPN structure [74] and the proposal of SKINNY and MIDORI. The maximum number of attacked rounds can be evaluated considering the maximum length of three features: partial-matching, initial structure and splice-and-cut. For partial-matching, the number of rounds in both forward and backward directions cannot reach to the full diffusion rounds (which for CRAFT is 7 rounds). Due to the key length being higher than the state size, we can add one more round in each direction. Also, because of the last linear half-round, partial-matching can work up to $2\times(7-1+1)+1 = 15$ rounds. In fact, there are four partial-matching characteristics for 15 rounds of CRAFT and Figure 6 depicts one of them.

The condition for the initial structure [75] is that the key differential trails in both forward and backward directions do not share active non-linear components. As any key differential in CRAFT affects all 16 Sboxes after at least $7 + 1$ rounds in both directions, there is no such differential which shares active Sbox(es) in more than 7 rounds. Therefore, it works up to 7 rounds for CRAFT.

Splice-and-cut may extend the number of attacked rounds up to the number of full diffusion rounds, i.e., 7. Thus, we conclude that *at most* a $(15 + 7 + 7) = 29$-round meet-in-the-middle attack might be feasible, but a higher number of rounds is secure. Note that the freedom of the tweak is already considered in the number of rounds for which there is a matching characteristic.

5.7 Integral Attack

Integral attacks [29, 55] are likely to be efficient for SPN block ciphers. The *integral attack* takes a set of plaintexts, in which particular cells are fixed to a constant value, and the other cells can contain all possible values. In case of CRAFT, each of such nibbles takes values in $\mathbb{F}_2^4$. Considering a set of such plaintexts, each cell of the cipher state belongs to one of the four cases below:

- All (**A**): all possible values in $\mathbb{F}_2^4$ appear the same number of times, i.e., a uniform distribution.
- Balanced (**B**): the XOR sum of all values in the cell is 0.
- Constant (**C**): the value of the cell is constant.
- Unknown (**U**): no particular property holds.

In order to find the longest integral characteristic for CRAFT, first we set one active nibble to the state, i.e., belonging to the **A** case. The cipher state is processed by the round functions until all nibbles become unknown, i.e., **U** case. Then, we extend it to a higher-order integral by propagating the active cell in the backward direction until all the nibbles become active. The longest integral characteristics that we found in this way covers 13 rounds, one of which is shown in Figure 7. We also checked the existence of integral distinguishers based on the *division property* [84, 86] using the Mixed-Integer Linear Programming approach described in [88]. With that, we did not find distinguishers for more than 13 rounds. By appending 7 rounds for key recovery to the rest of the characteristic, the attacker might have a successful attack on *at most* 20 rounds.

5.8 Invariant Attacks

In *invariant attacks* [85] the adversary aims at finding a (non-trivial) Boolean function g for which there exist many keys k (so-called *weak keys*), such that $g + g \circ \mathcal{E}_k$ is a

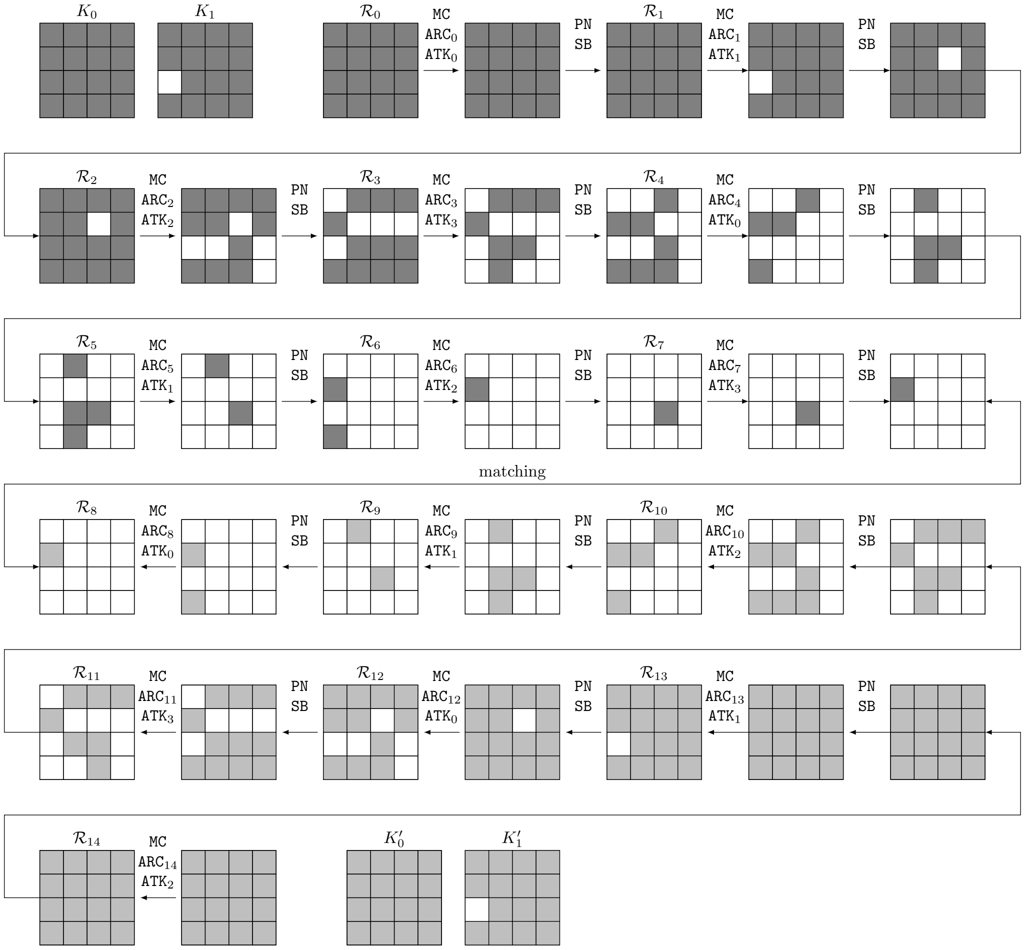


Figure 6: A 15-round partial matching meet-in-the-middle for `CRAFT`.

constant function. Such a g is called an *invariant* for $\mathcal{E}_k$. The knowledge of a non-trivial invariant could be used as a distinguisher on the cipher. The most promising approach for the adversary is to search for a Boolean function that is an invariant for *all* of the building blocks, i.e., the Sbox layer and the linear layer (plus key addition) in each round, simultaneously. Such an invariant could then be iterated over the rounds in order to cover the full cipher. For SPNs with a simple key schedule, [11] presents a security argument on the resistance against invariant attacks based on the round constants of the cipher. It can easily be applied to `CRAFT`. In particular, let $\mathcal{R}_i$ and $\mathcal{R}_j$ be two rounds in which the same round tweakey is added and let $d_{i,j} := \texttt{PN} \circ \texttt{MC}(RC_i) \oplus \texttt{PN} \circ \texttt{MC}(RC_j)$. If the smallest $(\texttt{PN} \circ \texttt{MC})$-invariant subspace that contains $d_{i,j}$ has a dimension of at least $n-1 = 63$, then any Boolean function that is an invariant for both $\texttt{PN} \circ \texttt{MC} \circ \texttt{ARC}_i \circ \texttt{ATK}'_i$ and $\texttt{PN} \circ \texttt{MC} \circ \texttt{ARC}_j \circ \texttt{ATK}'_j$ must be trivial or affine. Note that here for the simplicity, we re-ordered the round operations and considered an equivalent tweakey which is used before the `MC` operation. However, as the Sbox has no component of algebraic degree one, such an invariant would be useless in order to attack the cipher. For `CRAFT`, the smallest $\texttt{PN} \circ \texttt{MC}$-invariant subspace that contains $\{d_{0,4}, d_{1,5}, d_{2,6}, d_{3,7}\}$ is $\mathbb{F}_2^{64}$, i.e., it has dimension 64. Therefore, we already get a sound security argument on the resistance against invariant attacks after 8 rounds.

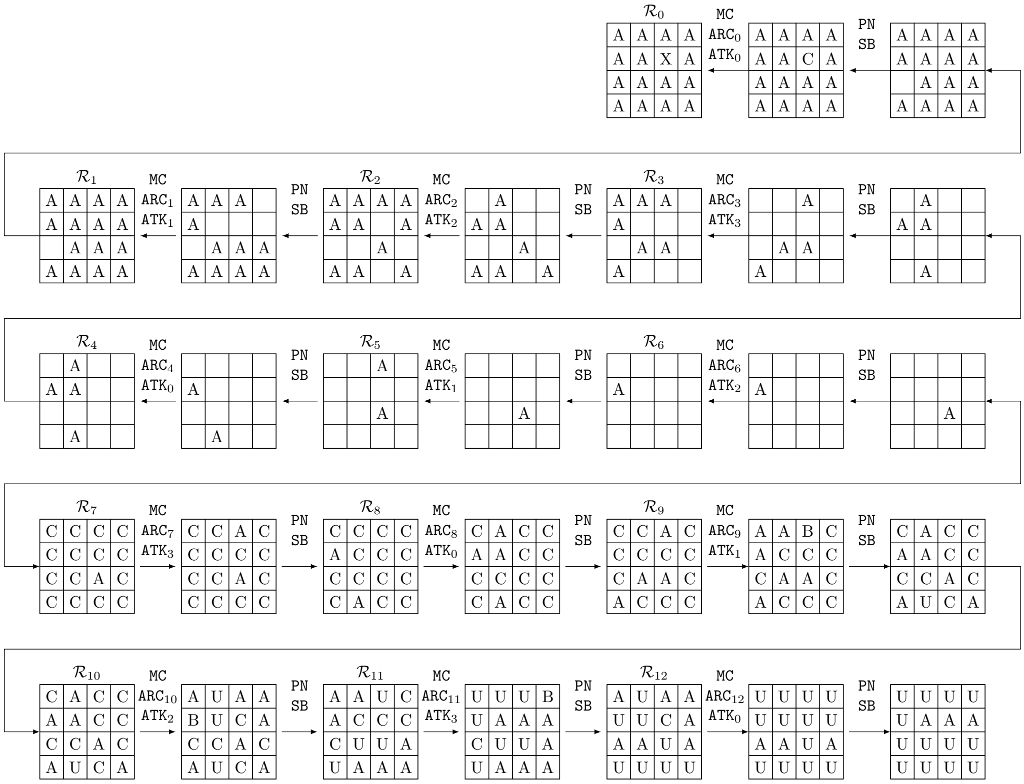


Figure 7: A 13-round integral distinguisher for CRAFT.

6 Hardware Implementations

As stated before, our target is a round-based implementation, where one round of the cipher is completed at every clock cycle. This leads to the design architecture shown in Figure 8, which supports both encryption and decryption functionalities. To this end, we just need to add a MC module through the round key path and a multiplexer to decide whether the selected round key TK_i or $\texttt{MC}(TK_i)$ should be given to the round function. It is noteworthy that since MC does not change the third and fourth rows (see Section 3), 32 bits of TK_i and $\texttt{MC}(TK_i)$ are the same. This help us to save 32 multiplexers. In total, supporting decryption (only with respect to the round key) costs $3 \times 16 = 48$ XOR gates and 32 multiplexers. To provide the round constant for decryption, we implemented the update function of each LFSR with both forward and backward functionalities, selected by the encryption/decryption E/D signal bit. Hence, we further need to initialize the LFSRs with either (1,1) or (8,5) for encryption/decryption respectively (see Table 2). As given in Section 4.4, each LFSR for either forward or backward needs only one XOR gate for the feedback function. This means that supporting decryption needs 2 extra XOR gates and 7 multiplexers. Note that different initial values for encryption/decryption can be generated by the E/D signal and only one NOT gate. More precisely, $\texttt{\=E00E}, \texttt{\=E01}$ generates $\texttt{0001}, \texttt{001}$ for encryption ($\texttt{E} = \texttt{1}$) and $\texttt{1000}, \texttt{101}$ for decryption ($\texttt{E} = \texttt{0}$). In total, turning an encryption-only implementation of CRAFT into encryption/decryption costs at most 50 XOR gates, 39 multiplexers, and 1 inverter. The hardware synthesizers usually merge the logic and even achieve a smaller overhead. Independent of whether decryption is supported by an implementation, adding the tweak support needs 64 multiplexers (to choose T or $Q(T)$) and 64 XOR gates (to add it to the K_i).

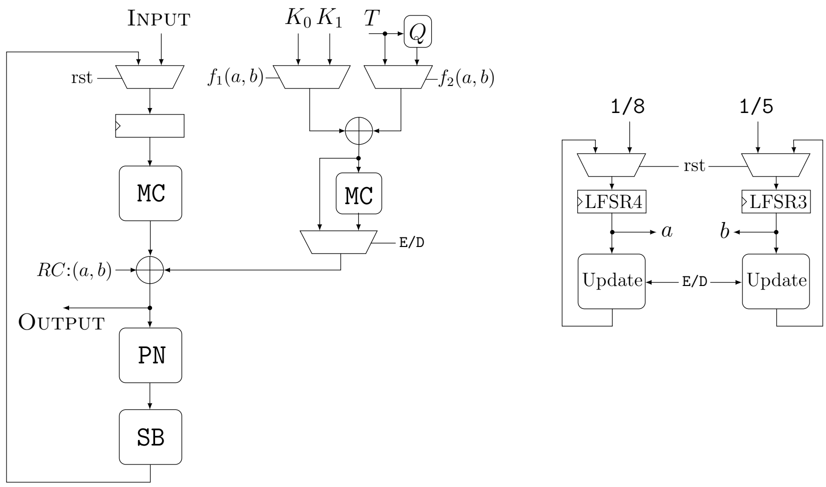


Figure 8: Round-based design architecture of `CRAFT` supporting tweak, encryption and decryption

For the implementations we used Synopsys Design Compiler with the IBM 130 nm ASIC standard cell library. The result of pure implementations (not protected against either SCA or DFA attacks) are shown in the first column of Table 6. Surprisingly, the only-encryption `CRAFT` without tweak needs less than 1000 GE which – to the best of our knowledge – is a record for a round-based implementation with 64-bit state and 128-bit key. We should highlight that due to the key-alternating fashion of the key schedule, we do not need to use registers dedicated to the key state. Instead, we have to use large multiplexers (see Figure 8) to select the corresponding 64-bit round key. The same approach has been used in the design and implementation of `MIDORI` [7], `PICCOLO` [80], and `KTANTAN` [24]. We also did not use register for the key state in the implementations of these three ciphers, but under the same condition `CRAFT` outperforms `MIDORI` and `PICCOLO` with a large distance (see Table 6). It is noteworthy that we have implemented all considered ciphers ourselves following a unique design architecture and implementation fashion allowing us 1) to synthesize all of them under the same ASIC library[10], and 2) to apply the underlying countermeasure to DFA attacks enabling a fair comparison. We further have not used any extraordinary scan flip-flops, as an optimized combination of a flip-flop and a multiplexer. Therefore, the area footprints given in Table 6 do not necessarily fit to the numbers reported in original documents each of which synthesized by a different library.

In fact, serialized architectures (e.g. nibble-serial) have been used in several lightweight ciphers to achieve a low area footprint but with a high latency. As an example, a bit-serial implementation of `SIMON` with area footprint of 958 GE needs 2816 clock cycles [10]. Serializing `CRAFT` would need extra registers for the key state to provide the key bits/nibbles/bytes per clock cycle. This implies that a serialized `CRAFT` with high latency needs more area compared to its round-based variant accomplishing the encryption/decryption in 32 clock cycles. Notably, the critical path delay (inverse of maximum clock frequency) of `CRAFT` is higher than that of `PRESENT`, `GIFT`, and `SIMON`. This is because in such ciphers the diffusion layer is realized by a bit permutation, which induces no delay at all. In our comparisons, we also included `KATAN` and `KTANTAN` with 64-bit state, although their 80-bit key size and the high number of 762 clock cycles per encryption do not match the other ciphers considered. We further included `SKINNY` with a 192-bit tweakey which is compatible to `CRAFT` supporting a 64-bit tweak.

As a side note, it can be seen that `CRAFT` is smaller than `MIDORI` while they share some

[10] Synthesizing a single design using different ASIC libraries can lead to very diverse results [44].

Table 6: Area (GE) and Latency (ns) comparison of round-based implementations considering an $\mathcal{M}_{t=d-1}$-bounded univariate adversary with an $[n, k, d]$ code, using the IBM 130 nm ASIC library, partially borrowed from [1].

Algorithm	Key	clock cycles	unprotected area	unprotected latency	$[\mathbf{5,4,2}]$ area	$[\mathbf{5,4,2}]$ latency	$[\mathbf{6,4,2}]$ area	$[\mathbf{6,4,2}]$ latency	$[\mathbf{7,4,3}]$ area	$[\mathbf{7,4,3}]$ latency	$[\mathbf{8,4,4}]$ area	$[\mathbf{8,4,4}]$ latency
SKINNY Enc	128	37	1738	3.66	3640	5.16	4494	5.24	5636	6.09	6804	6.37
LED Enc	128	49	1664	9.15	4499	9.57	5264	9.17	6699	10.04	8718	12.80
MIDORI Enc	128	17	1372	7.57	3282	8.25	3942	8.16	5262	8.87	6840	10.40
PRESENT Enc	128	32	1767	2.93	4211	5.19	5177	5.62	6639	6.32	8219	6.71
GIFT Enc	128	29	1587	2.88	3824	5.11	4722	5.32	6082	6.11	7767	6.61
SIMON Enc	128	45	1629	2.86	3614	5.20	4487	5.27	5621	5.93	7603	6.44
PICCOLO Enc	128	32	1462	7.69	3870	9.86	4763	9.47	6241	10.10	8217	12.15
KATAN Enc	80	762	1080	3.87	2946	5.84	3610	6.04	4746	6.69	6684	7.49
KTANTAN Enc	80	762	601	4.23	2039	5.38	2457	5.07	3069	5.47	4207	6.01
CRAFT Enc	128	32	**949**	3.19	**2342**	5.13	**2857**	5.25	**3698**	5.56	**5014**	6.26
CRAFT Enc&Dec	128	32	**1089**	3.60	**2609**	5.07	**3169**	5.40	**4069**	6.19	**5459**	6.43
CRAFT Enc Tweak	128	32	**1193**	3.37	**2801**	5.15	**3420**	5.27	**4518**	5.56	**6657**	6.25
CRAFT Enc&Dec T.	128	32	**1339**	3.99	**3066**	5.36	**3731**	5.43	**4891**	6.09	**7110**	6.64
SKINNY Enc	192	41	2206	4.00	4540	5.63	5656	5.74	7119	6.34	8553	6.74

components. At the same time, CRAFT needs a higher number of clock cycles (32 versus 17). However, the whole latency (of the entire encryption) of the slowest CRAFT (with tweak and decryption support) is smaller than that of MIDORI (127.68 ns versus 128.69 ns). Of course, due to higher number of clock cycles, CRAFT cannot outperform MIDORI with respect to energy consumption per encryption.

Note that we have not restricted the clock period in our syntheses allowing the synthesizer to achieve the smallest possible area. However, it is possible to force the synthesizer to reach a certain maximum latency which leads to higher area requirement. As a reference, in Figure 9 we show such results for round-based implementations of CRAFT. We used the IBM 130 nm ASIC standard cell library due to its public availability and the fact that it is used to benchmark SIMON area footprints in [10]. In order to give an overview on the performance comparisons under a more modern ASIC library, we repeated all our syntheses using a commercial 40 nm standard cell library[11]. The corresponding results are shown in Appendix D.

6.1 Protection against DFA Attacks

Adversary Model. As stated before, we focus on the fault-detection technique proposed in [1], where two adversary models are defined:

- Univariate model $\mathcal{M}_t$, where at only one clock cycle of each encryption process the adversary is able to make at most t cells of the entire circuit faulty.

[11] Due to an NDA, the full name of the used library is omitted.

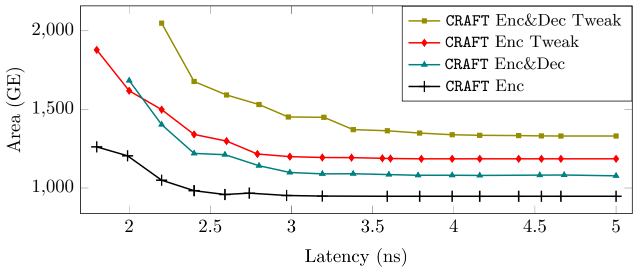


Figure 9: Latency versus area of unprotected round-based CRAFT, using the IBM 130 nm ASIC library.

- Multivariate adversary $\mathcal{M}_t^*$, which is bounded to t faulty cells in the entire circuit at every clock cycle.

This implies that the safe-error [89] and stuck-at-0/1 [26] models are not covered. Further, our fault-protected implementations do not necessarily provide security against FSA [60] and SIFA [34]. Protection against such kind of attacks needs either a clock glitch detector [36] or a combination of different countermeasures, e.g. a fault-correction technique.

Results. We considered four cases for the redundancy size $m \in \{1, \ldots, 4\}$ bits. The corresponding design architectures for $m < k = 4$ and $m \geq k = 4$ are shown in Figure 10 and Figure 11 respectively (in Appendix C). We further considered $[l, k, d]$ codes $[5, 4, 2]$, $[6, 4, 2]$, $[7, 4, 3]$, and $[8, 4, 4]$ for $m = 1, 2, 3, 4$ respectively. Generator matrices of these codes have been given in Section 4.2. This implies that with $m = 1$ and $m = 2$ (both leading to $d = 2$) the circuit is able to detect at most $t = d - 1 = 1$ faulty cell, i.e., protection against an $\mathcal{M}_{t=1}$ adversary. This is improved by larger $m = 3$ and $m = 4$ to protect against an $\mathcal{M}_{t=2}$ and $\mathcal{M}_{t=3}$ adversary respectively. Note that MC of CRAFT has been chosen to 1) let MC operate solely on the redundant part of information for $m < k$ (see Figure 10), and 2) avoid any necessary extra check point at MC input (see [1, Lemma 4 and Theorem 1]). This, in addition to the LFSRs with at most 4-bit width (since $k = 4$), help us to realize such implementations with low area overhead. The performance figure and area requirement of several implementations compared to that of other ciphers are listed in Table 6. It can be seen that CRAFT outperforms all other considered ciphers with compatible state and key size even when CRAFT supports both encryption and decryption.

It might be thought that the fault-protected implementations of CRAFT are smaller than the others since its unprotected variant is smaller. Table 6 shows the inconsistency of this statement. As an example, unprotected LED and GIFT need less area compared to SKINNY, while their fault-protected variants are larger.

According to [1], to provide security against a multivariate adversary $\mathcal{M}_t^*$, extra check points should be defined and the consistency check module needs to be adjusted. Doing so, we achieved again the smallest area overhead for CRAFT under all considered settings. The results are given in Table 8 (in Appendix C). As stated, the synthesis results using a commercial 40 nm ASIC library are given in Appendix D.

Experiments. Since ASIC fabrication which enables practical experiments is time consuming, we have conducted a few simulations to ensure the fault-detection capability of our implementations. 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

Table 7: Area (GE) and Latency (ns) of round-based **Threshold Implementations** of CRAFT with 3 shares considering an $\mathcal{M}_{t=d-1}$-bounded univariate adversary with an $[n, k, d]$ code, using the IBM 130 nm ASIC library.

Algorithm	Key	clock cycles	only TI area	only TI latency	TI+[5,4,2] area	TI+[5,4,2] latency	TI+[6,4,2] area	TI+[6,4,2] latency	TI+[7,4,3] area	TI+[7,4,3] latency	TI+[8,4,4] area	TI+[8,4,4] latency
CRAFT Enc	128	64	5106	4.05	10620	5.63	13851	5.80	16049	6.75	21687	6.99
CRAFT Enc&Dec	128	64	5303	4.33	10909	6.46	14194	6.28	16454	7.85	22210	7.20
CRAFT Enc Tweak	128	64	5412	4.17	11079	5.63	14416	5.80	16786	6.74	23351	6.95
CRAFT Enc&Dec T.	128	64	5605	4.92	11374	6.37	14763	6.27	17208	7.89	23868	7.16

CRAFT only-encryption (without tweak) protected against a multivariate adversary $\mathcal{M}^*_{t=1}$ with redundancy size $m = 1$ contains 1437 cells, i.e., a vector of 1437 signals to inject faults. Our simulations under the considered adversary model (i.e., single-bit faults at every clock cycle) showed 100% fault coverage.

6.2 Combined Protection against SCA and DFA Attacks

Application of masking as the most common countermeasure against SCA attacks is challenging when the underlying function is not transparent to the applied masking scheme. In the most common technique, i.e., Boolean masking, the difficulty of realizing a masked implementation is summarized to providing a secure masked variant of its non-linear functions while linear operations can be repeated with respect to the order of the employed masking scheme. Threshold Implementation (TI) [68] formalized this process and defined requirements to be fulfilled for a provably-secure implementation (up to a certain order). Hence, in order to realize a TI variant of CRAFT, we need to just provide its TI Sbox; masked version of its other operations are straightforwardly made due to their linearity. CRAFT's Sbox belongs to the cubic class 266 [18], and it has been shown that such a class can be uniformly shared in two stages with minimum number of 3 shares. This means that we need to put a register in the middle of the TI Sbox. CRAFT's Sbox is the same as that of MIDORI, and a correct and uniform TI of MIDORI's Sbox with 3 shares in two stages is given in [66]. We have taken such a design for the Sbox and easily repeated the other modules 3 times to realize a first-order secure round-based 3-share TI of CRAFT without any fresh randomness. The design architecture is very similar to the one shown in Figure 8, but with a register stage in SB module. Therefore, the entire encryption/decryption takes now 64 clock cycles, but it forms a pipeline, where in 64 clock cycles two e.g. encryptions can be accomplished. The performance figures of such implementations are given in the first column of Table 7.

We further applied the same fault-detection mechanism explained before on such implementations using all four considered codes. This offers protection against first-order SCA attacks as well as a univariate fault-injection adversary model $\mathcal{M}_t$ with $t = 1, 2, 3$. The rest of Table 7 shows the corresponding results. It is noteworthy that the result for the similar implementations considering the corresponding multivariate adversary is shown in Table 9 (in Appendix C), and by a commercial 40 nm ASIC library in Appendix D.

7 Conclusions

This paper introduced the block cipher CRAFT, for which the resistance of its implementations against DFA attacks was taken into account during the design phase. Considering one of the recent developments in the areas of fault detection, we have designed the building blocks of CRAFT leading to very limited area overhead. For the unprotected implementation as well as the one equipped with fault-detection mechanisms, the corresponding results show a clear distance between CRAFT and the state of the art. To the best of our knowledge, it is a unique construction with 128-bit key whose round-based implementation (requiring 32 clock cycles to encrypt a 64-bit message) needs less than 1000 GE. Further, it offers two other interesting features by (a) supporting a 64-bit tweak which adds around 245 GE area and (b) being able to turn into decryption function with a very low area overhead of around 140 GE.

For further protection against the attacks such as SIFA [34], an interesting topic for future work is to add error-correction capabilities. Since our underlying fault-detection scheme is based on application of binary linear codes, it might be promising to adjust the same principle to correct faults (of course using the codes with larger distance).

References

[1] Anita Aghaie, Amir Moradi, Shahram Rasoolzadeh, Falk Schellenberg, and Tobias Schneider. Impeccable Circuits. *IACR Cryptology ePrint Archive*, 2018:203, 2018.

[2] Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, and Assia Tria. How to flip a bit? In *IOLTS 2010*, pages 235–239. IEEE Computer Society, 2010.

[3] Michel Agoyan, Jean-Max Dutertre, David Naccache, Bruno Robisson, and Assia Tria. When Clocks Fail: On Critical Paths and Clock Faults. In Dieter Gollmann, Jean-Louis Lanet, and Julien Iguchi-Cartigny, editors, *CARDIS 2010*, volume 6035 of *LNCS*, pages 182–193. Springer, 2010.

[4] Martin R. Albrecht, Benedikt Driessen, Elif Bilge Kavun, Gregor Leander, Christof Paar, and Tolga Yalçin. Block Ciphers - Focus on the Linear Layer (feat. PRIDE). In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014*, volume 8616 of *LNCS*, pages 57–76. Springer, 2014.

[5] Ralph Ankele, Subhadeep Banik, Avik Chakraborti, Eik List, Florian Mendel, Siang Meng Sim, and Gaoli Wang. Related-Key Impossible-Differential Attack on Reduced-Round Skinny. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 2017*, volume 10355 of *LNCS*, pages 208–228. Springer, 2017.

[6] Roberto Avanzi. The QARMA Block Cipher Family. Almost MDS Matrices Over Rings With Zero Divisors, Nearly Symmetric Even-Mansour Constructions With Non-Involutory Central Rounds, and Search Heuristics for Low-Latency S-Boxes. *IACR Trans. Symmetric Cryptol.*, 2017(1):4–44, 2017.

[7] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A Block Cipher for Low Energy. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015*, volume 9453 of *LNCS*, pages 411–436. Springer, 2015.

[8] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A Small Present - Towards Reaching the Limit of Lightweight Encryption. In Fischer and Homma [37], pages 321–345.

[9] Elad Barkan and Eli Biham. In How Many Ways Can You Write Rijndael? In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 160–175. Springer, 2002.

[10] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK lightweight block ciphers. In *DAC 2015*, pages 175:1–175:6, 2015.

[11] Christof Beierle, Anne Canteaut, Gregor Leander, and Yann Rotella. Proving Resistance Against Invariant Attacks: How to Choose the Round Constants. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017*, volume 10402 of *LNCS*, pages 647–678. Springer, 2017.

[12] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In Robshaw and Katz [73], pages 123–153.

[13] Guido Bertoni, Luca Breveglieri, Israel Koren, Paolo Maistri, and Vincenzo Piuri. Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard. *IEEE Trans. Computers*, 52(4):492–505, 2003.

[14] Guido Bertoni and Jean-Sébastien Coron, editors. *CHES 2013*, volume 8086 of *LNCS*. Springer, 2013.

[15] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In Jacques Stern, editor, *EUROCRYPT 1999*, volume 1592 of *LNCS*, pages 12–23. Springer, 1999.

[16] Eli Biham and Adi Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In Burton S. Kaliski Jr., editor, *CRYPTO 1997*, volume 1294 of *LNCS*, pages 513–525. Springer, 1997.

[17] Begül Bilgin, Andrey Bogdanov, Miroslav Knezevic, Florian Mendel, and Qingju Wang. Fides: Lightweight Authenticated Cipher with Side-Channel Resistance for Constrained Hardware. In Bertoni and Coron [14], pages 142–158.

[18] Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, Natalia N. Tokareva, and Valeriya Vitkup. Threshold implementations of small S-boxes. *Cryptography and Communications*, 7(1):3–33, 2015.

[19] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In Paillier and Verbauwhede [70], pages 450–466.

[20] Andrey Bogdanov and Vincent Rijmen. Linear Hulls with Correlation Zero and Linear Cryptanalysis of Block Ciphers. *Des. Codes Cryptography*, 70(3):369–383, 2014.

[21] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 208–225. Springer, 2012.

[22] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, 2004.

[23] Gaetan Canivet, Paolo Maistri, Régis Leveugle, Jessy Clédière, Florent Valette, and Marc Renaudin. Glitch and Laser Fault Attacks onto a Secure AES Implementation on a SRAM-Based FPGA. *J. Cryptology*, 24(2):247–268, 2011.

[24] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In Christophe Clavier and Kris Gaj, editors, *CHES 2009*, volume 5747 of *LNCS*, pages 272–288. Springer, 2009.

[25] Jung Hee Cheon and Tsuyoshi Takagi, editors. *ASIACRYPT 2016*, volume 10031 of *LNCS*, 2016.

[26] Christophe Clavier. Secret External Encodings Do Not Prevent Transient Fault Analysis. In Paillier and Verbauwhede [70], pages 181–194.

[27] Tingting Cui, Keting Jia, Kai Fu, Shiyao Chen, and Meiqin Wang. New Automatic Search Tool for Impossible Differentials and Zero-Correlation Linear Approximations. *IACR Cryptology ePrint Archive*, 2016:689, 2016.

[28] Joan Daemen. *Cipher and Hash Function Design, Strategies Based on Linear and Differential Cryptanalysis, PhD Thesis.* K.U.Leuven, 1995.

[29] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher square. In Eli Biham, editor, *FSE 1997*, pages 149–165, 1997.

[30] Joan Daemen, Michaël Peeters, Gilles Van Assche, and Vincent Rijmen. Nessie Proposal: NOEKEON. In *First Open NESSIE Workshop*, pages 213–230, 2000.

[31] Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, and Assia Tria. Electromagnetic Transient Faults Injection on a Hardware and a Software Implementations of AES. In Guido Bertoni and Benedikt Gierlichs, editors, *FDTC 2012*, pages 7–15. IEEE Computer Society, 2012.

[32] Whitfield Diffie and Martin E. Hellman. Special Feature Exhaustive Cryptanalysis of the NBS Data Encryption Standard. *IEEE Computer*, 10(6):74–84, 1977.

[33] Christoph Dobraunig, Maria Eichlseder, Hannes Groß, Stefan Mangard, Florian Mendel, and Robert Primas. Statistical Ineffective Fault Attacks on Masked AES with Fault Countermeasures. In Thomas Peyrin and Steven D. Galbraith, editors, *ASIACRYPT 2018*, volume 11273 of *LNCS*, pages 315–342. Springer, 2018.

[34] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. SIFA: Exploiting Ineffective Fault Inductions on Symmetric Cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):547–572, 2018.

[35] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Markus Schofnegger. Algebraic Cryptanalysis of Frit. *IACR Cryptology ePrint Archive*, 2018:809, 2018.

[36] Sho Endo, Yang Li, Naofumi Homma, Kazuo Sakiyama, Kazuo Ohta, Daisuke Fujimoto, Makoto Nagata, Toshihiro Katashita, Jean-Luc Danger, and Takafumi Aoki. A Silicon-Level Countermeasure Against Fault Sensitivity Analysis and Its Evaluation. *IEEE Trans. VLSI Syst.*, 23(8):1429–1438, 2015.

[37] Wieland Fischer and Naofumi Homma, editors. *CHES 2017*, volume 10529 of *LNCS*. Springer, 2017.

[38] Benoît Gérard, Vincent Grosso, María Naya-Plasencia, and François-Xavier Standaert. Block Ciphers That Are Easier to Mask: How Far Can We Go? In Bertoni and Coron [14], pages 383–399.

[39] Benedikt Gierlichs, Jörn-Marc Schmidt, and Michael Tunstall. Infective Computation and Dummy Rounds: Fault Protection for Block Ciphers without Check-before-Output. In Alejandro Hevia and Gregory Neven, editors, *LATINCRYPT 2012*, volume 7533 of *LNCS*, pages 305–321. Springer, 2012.

[40] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations. In Carlos Cid and Christian Rechberger, editors, *FSE 2014*, volume 8540 of *LNCS*, pages 18–37. Springer, 2014.

[41] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED Block Cipher. In Preneel and Takagi [72], pages 326–341.

[42] Xiaofei Guo and Ramesh Karri. Invariance-based Concurrent Error Detection for Advanced Encryption Standard. In *DAC 2012*, pages 573–578. ACM, 2012.

[43] Xiaofei Guo, Debdeep Mukhopadhyay, Chenglu Jin, and Ramesh Karri. Security Analysis of Concurrent Error Detection against Differential Fault Analysis. *J. Cryptographic Engineering*, 5(3):153–169, 2015.

[44] Jérémy Jean, Amir Moradi, Thomas Peyrin, and Pascal Sasdrich. Bit-Sliding: A Generic Technique for Bit-Serial Implementations of SPN-based Primitives - Applications to AES, PRESENT and SKINNY. In Fischer and Homma [37], pages 687–707.

[45] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and Keys for Block Ciphers: The TWEAKEY Framework. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014*, volume 8874 of *LNCS*, pages 274–288. Springer, 2014.

[46] Marc Joye, Pascal Manet, and Jean-Baptiste Rigaud. Strengthening hardware AES implementations against fault attacks. *IET Information Security*, 1(3):106–110, 2007.

[47] Mark G. Karpovsky, Konrad J. Kulikowski, and Alexander Taubin. Robust Protection against Fault-Injection Attacks on Smart Cards Implementing the Advanced Encryption Standard. In *DSN 2004*, pages 93–101. IEEE Computer Society, 2004.

[48] Ramesh Karri, Kaijie Wu, Piyush Mishra, and Yongkook Kim. Concurrent Error Detection Schemes for Fault-based Side-Channel Cryptanalysis of Symmetric Block Ciphers. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 21(12):1509–1517, 2002.

[49] John Kelsey, Bruce Schneier, and David A. Wagner. Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In Koblitz [56], pages 237–251.

[50] Mehran Mozaffari Kermani and Arash Reyhani-Masoleh. Parity-Based Fault Detection Architecture of S-box for Advanced Encryption Standard. In *DFT 2006*, pages 572–580. IEEE Computer Society, 2006.

[51] Mehran Mozaffari Kermani and Arash Reyhani-Masoleh. A Lightweight Concurrent Fault Detection Scheme for the AES S-Boxes Using Normal Basis. In Oswald and Rohatgi [69], pages 113–129.

[52] Mehran Mozaffari Kermani and Arash Reyhani-Masoleh. Concurrent Structure-Independent Fault Detection Schemes for the Advanced Encryption Standard. *IEEE Trans. Computers*, 59(5):608–622, 2010.

[53] Mehran Mozaffari Kermani and Arash Reyhani-Masoleh. A Lightweight High-Performance Fault Detection Scheme for the Advanced Encryption Standard Using Composite Fields. *IEEE Trans. VLSI Syst.*, 19(1):85–91, 2011.

[54] Lars Knudsen. DEAL - A 128-bit Block Cipher. In *NIST AES Proposal*, 1998.

[55] Lars R. Knudsen and David A. Wagner. Integral Cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *FSE 2002*, volume 2365 of *LNCS*, pages 112–127. Springer, 2002.

[56] Neal Koblitz, editor. *CRYPTO 1996*, volume 1109 of *LNCS*. Springer, 1996.

[57] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Koblitz [56], pages 104–113.

[58] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael J. Wiener, editor, *CRYPTO 1999*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.

[59] Thorsten Kranz, Gregor Leander, and Friedrich Wiemer. Linear Cryptanalysis: Key Schedules and Tweakable Block Ciphers. *IACR Trans. Symmetric Cryptol.*, 2017(1):474–505, 2017.

[60] Yang Li, Kazuo Sakiyama, Shigeto Gomisawa, Toshinori Fukunaga, Junko Takahashi, and Kazuo Ohta. Fault Sensitivity Analysis. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 320–334. Springer, 2010.

[61] Moses Liskov, Ronald L. Rivest, and David A. Wagner. Tweakable Block Ciphers. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 31–46. Springer, 2002.

[62] Guozhen Liu, Mohona Ghosh, and Ling Song. Security Analysis of SKINNY under Related-Tweakey Settings (Long Paper). *IACR Trans. Symmetric Cryptol.*, 2017(3):37–72, 2017.

[63] Paolo Maistri and Régis Leveugle. Double-Data-Rate Computation as a Countermeasure against Fault Analysis. *IEEE Trans. Computers*, 57(11):1528–1539, 2008.

[64] Tal Malkin, François-Xavier Standaert, and Moti Yung. A Comparative Cost/Security Analysis of Fault Attack Countermeasures. In Luca Breveglieri, Israel Koren, David Naccache, and Jean-Pierre Seifert, editors, *FDTC 2006*, volume 4236 of *LNCS*, pages 159–172. Springer, 2006.

[65] Mitsuru Matsui. On Correlation Between the Order of S-boxes and the Strength of DES. In Alfredo De Santis, editor, *EUROCRYPT 1994*, volume 950 of *LNCS*, pages 366–375. Springer, 1994.

[66] Amir Moradi and Tobias Schneider. Side-Channel Analysis Protection and Low-Latency in Action - - Case Study of PRINCE and Midori. In Cheon and Takagi [25], pages 517–547.

[67] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In Chuankun Wu, Moti Yung, and Dongdai Lin, editors, *Inscrypt 2011*, volume 7537 of *LNCS*, pages 57–76. Springer, 2011.

[68] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. *J. Cryptology*, 24(2):292–321, 2011.

[69] Elisabeth Oswald and Pankaj Rohatgi, editors. *CHES 2008*, volume 5154 of *LNCS*. Springer, 2008.

[70] Pascal Paillier and Ingrid Verbauwhede, editors. *CHES 2007*, volume 4727 of *LNCS*. Springer, 2007.

[71] Gilles Piret, Thomas Roche, and Claude Carlet. PICARO - A Block Cipher Allowing Efficient Higher-Order Side-Channel Resistance. In Feng Bao, Pierangela Samarati, and Jianying Zhou, editors, *ACNS 2012*, volume 7341 of *LNCS*, pages 311–328. Springer, 2012.

[72] Bart Preneel and Tsuyoshi Takagi, editors. *CHES 2011*, volume 6917 of *LNCS*. Springer, 2011.

[73] Matthew Robshaw and Jonathan Katz, editors. *CRYPTO 2016*, volume 9815 of *LNCS*. Springer, 2016.

[74] Yu Sasaki. Meet-in-the-Middle Preimage Attacks on AES Hashing Modes and an Application to Whirlpool. In Antoine Joux, editor, *FSE 2011*, volume 6733 of *LNCS*, pages 378–396. Springer, 2011.

[75] Yu Sasaki and Kazumaro Aoki. Finding Preimages in Full MD5 Faster Than Exhaustive Search. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 134–152. Springer, 2009.

[76] Yu Sasaki and Yosuke Todo. New Impossible Differential Search Tool from Design and Cryptanalysis Aspects - Revealing Structural Properties of Several Ciphers. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017*, volume 10212 of *LNCS*, pages 185–215, 2017.

[77] Akashi Satoh, Takeshi Sugawara, Naofumi Homma, and Takafumi Aoki. High-Performance Concurrent Error Detection Scheme for AES Hardware. In Oswald and Rohatgi [69], pages 100–112.

[78] Tobias Schneider, Amir Moradi, and Tim Güneysu. ParTI - Towards Combined Hardware Countermeasures Against Side-Channel and Fault-Injection Attacks. In Robshaw and Katz [73], pages 302–332.

[79] Nidhal Selmane, Sylvain Guilley, and Jean-Luc Danger. Practical Setup Time Violation Attacks on AES. In *EDCC-7 2008*, pages 91–96. IEEE Computer Society, 2008.

[80] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: An Ultra-Lightweight Blockcipher. In Preneel and Takagi [72], pages 342–357.

[81] Thierry Simon, Lejla Batina, Joan Daemen, Vincent Grosso, Pedro Maat Costa Massolino, Kostas Papagiannopoulos, Francesco Regazzoni, and Niels Samwel. Towards Lightweight Cryptographic Primitives with Built-in Fault-Detection. *IACR Cryptology ePrint Archive*, 2018:729, 2018.

[82] François-Xavier Standaert, Gilles Piret, Gaël Rouvroy, Jean-Jacques Quisquater, and Jean-Didier Legat. ICEBERG : An Involutional Cipher Efficient for Block Encryption in Reconfigurable Hardware. In Bimal K. Roy and Willi Meier, editors, *FSE 2004*, volume 3017 of *LNCS*, pages 279–299. Springer, 2004.

[83] Siwei Sun, Lei Hu, Ling Song, Yonghong Xie, and Peng Wang. Automatic Security Evaluation of Block Ciphers with S-bP Structures Against Related-Key Differential Attacks. In Dongdai Lin, Shouhuai Xu, and Moti Yung, editors, *Inscrypt 2013*, volume 8567 of *LNCS*, pages 39–51. Springer, 2013.

[84] Yosuke Todo. Structural Evaluation by Generalized Integral Property. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 287–314. Springer, 2015.

[85] Yosuke Todo, Gregor Leander, and Yu Sasaki. Nonlinear Invariant Attack - Practical Attack on Full SCREAM, iSCREAM, and Midori64. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016*, volume 10032 of *LNCS*, pages 3–33, 2016.

[86] Yosuke Todo and Masakatu Morii. Bit-Based Division Property and Application to Simon Family. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 357–377. Springer, 2016.

[87] Kaijie Wu, Ramesh Karri, Grigori Kuznetsov, and Michael Gössel. Low Cost Concurrent Error Detection for the Advanced Encryption Standard. In *ITC 2004*, pages 1242–1248. IEEE Computer Society, 2004.

[88] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP Method to Searching Integral Distinguishers Based on Division Property for 6 Lightweight Block Ciphers. In Cheon and Takagi [25], pages 648–678.

[89] Sung-Ming Yen and Marc Joye. Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. *IEEE Trans. Computers*, 49(9):967–970, 2000.

[90] Sung-Ming Yen, Seungjoo Kim, Seongan Lim, and Sang-Jae Moon. RSA Speedup with Residue Number System Immune against Hardware Fault Cryptanalysis. In Kwangjo Kim, editor, *ICISC 2001*, volume 2288 of *LNCS*, pages 397–413. Springer, 2001.

A CRAFT Test Vectors

K_0	0000000000000000	0000000000000000	0123456789ABCDEF	27A6781A43F364BC
K_1	0000000000000000	0000000000000000	FEDCBA9876543210	916708D5FBB5AEFE
T	0000000000000000	0123456789ABCDEF	0000000000000000	54CD94FFD0670A58
TK_0	0000000000000000	0123456789ABCDEF	0123456789ABCDEF	736BECE593946EE4
TK_1	0000000000000000	0123456789ABCDEF	FEDCBA9876543210	C5AA9C2A2BD2A4A6
TK_2	0000000000000000	CAF5E892B374601D	0123456789ABCDEF	212225163E0A91F6
TK_3	0000000000000000	CAF5E892B374601D	FEDCBA9876543210	97E355D9864C5BB4
P	0000000000000000	0123456789ABCDEF	FEDCBA9876543210	5734F006D8D88A3E
$\mathcal{R}_1$	CCCCCCCCCAACCCCC	CCCCCCCC4026EEEE	6666666640265555	14E2E5E02782F597
$\mathcal{R}_2$	00001100CE8C1100	AD3C8902E34B3D7F	BF7E7F3D12C52098	ABA3B0CCB8C23B2C
$\mathcal{R}_3$	CAAC8400CDEC6802	DB2F3CB644A69625	7427F59408BD0CC8	11120F88604F9F6C
$\mathcal{R}_4$	DF8C4200C90D6EDF	86F3276DC7FA3B90	834244790FFE4848	80E2CF433C4292D7
$\mathcal{R}_5$	6F42C90221568B47	66F7B4EA96015517	78B1BF8BB62FB270	3603261F77DA2A0B
$\mathcal{R}_6$	785EBADFEB8BBB30	898616A193F1E0D1	C8CF7C05FC73D3D8	2841C0B837B44781
$\mathcal{R}_7$	C55385454DF684BD	08C08CDB427DCB33	7A432B785981FE74	72F759C4DA069BF7
$\mathcal{R}_8$	28E562EF6C580A8C	415DCA6974A1575E	E00F26DB942E3F1D	30CEE0B14FD1E4B1
$\mathcal{R}_9$	0C18B0F8F03343AE	A915C261B08C38AF	D6D682AB18CA1A6E	B81BE02B405B80B4
$\mathcal{R}_{10}$	4E313C63BABF68B5	C6BED93712A9CABD	4D8794F4BC82114D	DDEA85F954BB2B45
$\mathcal{R}_{11}$	BF8551562A8F0359	C1112A12A4D8EB62	D201DB392BA4AF54	351551FA14B133F8
$\mathcal{R}_{12}$	9C3B81D6CE56FB39	685717A0460DEB4E	E92E62BC68DB8D21	0F8E6D922A25B568
$\mathcal{R}_{13}$	9653B40F42869BE1	ADF1160F84E9E370	4EC07A4C62B1267C	0258595AA3063184
$\mathcal{R}_{14}$	A9548DEF40B463EE	6D49E2CDA3085E50	0AEF4EAB1B98E2A6	D9BD288E678EB7CB
$\mathcal{R}_{15}$	4F345CEEC62A1C48	234E7CA0CBF614EB	9D6E3D93F821D6D6	2DF389BEB583821D
$\mathcal{R}_{16}$	8A0EDF011EBFBBF9	F7E6887D2E99F1F3	F4E0748B2BE22EBE	9291033671E12848
$\mathcal{R}_{17}$	955654A66F486E8D	030A371D4459916D	A43BED19B29DC966	0EF17D4B9BE58B35
$\mathcal{R}_{18}$	2F48E6F8D675E939	DB08620DAF177A22	F6570E094527AA0D	3D693C57B6DE5852
$\mathcal{R}_{19}$	94937F2B09DACCEA	6A13F0A35F22E080	DF748000F7038B4A	E091288EA7F3D076
$\mathcal{R}_{20}$	100429C10EDAA13B	28C9B04F56FA6EEA	159BBA872FDD24AB	D8505AD6D13AC67C
$\mathcal{R}_{21}$	51A324C165E164B5	B13CB62A08BDA61A	E49E7F1FE8A5604A	81891DE4E3560725
$\mathcal{R}_{22}$	BFE54BFA7ADEC67B	BF56AA8FBE37458A	1BDB649A0E02D129	3138880E4745CBC1
$\mathcal{R}_{23}$	50F72174893A3ECC	7DB9E2C393574A48	FA0017892EEEB230	7B13E9762D290949
$\mathcal{R}_{24}$	0340398152E87CA4	BD1BDCD3ADA94B3F	C8CD58B1D2EA8CD9	2BD6F51B474F89C0
$\mathcal{R}_{25}$	E7014DB8DC4B2C0D	C8F2CEDD8775114B	FEA3E5BACE367229	E47D2E253B34F3D9
$\mathcal{R}_{26}$	2D0CE025575B7E7A	E20124C460EA646B	9EC3F85DC5650FC5	6B774CAF279F9314
$\mathcal{R}_{27}$	1747B7B5B976ED2C	FCE7932436E207BD	10DD00E46BB6BECB	DCD499AB12A24246
$\mathcal{R}_{28}$	042D795F99193A2E	CF719B8FC6857E2C	580242AD57675703	DA964E94327E523A
$\mathcal{R}_{29}$	431DA9997D3A7A11	3530D6E40A2EF657	091404200AD99EBB	43024A11EA859590
$\mathcal{R}_{30}$	A71A32718748E3FE	8355838B2ABE8F10	51018072CEA3311F	F3A3BA0717759526
$\mathcal{R}_{31}$	4436E78883063100	246C0991CF690C44	C606C7E8CF416941	CCE279D638A7C2DB
C	F630538883063100	2DB468477C1D6C59	9EDA9131B9155B51	A17D6BD4BEEB996F

B CRAFT C++ Code

```
const int S[16] =
  {0xc,0xa,0xd,0x3,0xe,0xb,0xf,0x7,0x8,0x9,0x1,0x5,0x0,0x2,0x4,0x6};
const int P[16] =
  {0xf,0xc,0xd,0xe,0xa,0x9,0x8,0xb,0x6,0x5,0x4,0x7,0x1,0x2,0x3,0x0};
const int Q[16] =
  {0xc,0xa,0xf,0x5,0xe,0x8,0x9,0x2,0xb,0x3,0x7,0x4,0x6,0x0,0x1,0xd};
const int RC3[32] =
  {0x1,0x4,0x2,0x5,0x6,0x7,0x3,0x1,0x4,0x2,0x5,0x6,0x7,0x3,0x1,0x4,
   0x2,0x5,0x6,0x7,0x3,0x1,0x4,0x2,0x5,0x6,0x7,0x3,0x1,0x4,0x2,0x5};
const int RC4[32] =
  {0x1,0x8,0x4,0x2,0x9,0xc,0x6,0xb,0x5,0xa,0xd,0xe,0xf,0x7,0x3,0x1,
   0x8,0x4,0x2,0x9,0xc,0x6,0xb,0x5,0xa,0xd,0xe,0xf,0x7,0x3,0x1,0x8};

const bool dec = 0;  // encryption:0 , decryption:1

int Key[2][16] = {
  {0x2,0x7,0xa,0x6,0x7,0x8,0x1,0xa,0x4,0x3,0xf,0x3,0x6,0x4,0xb,0xc},
  {0x9,0x1,0x6,0x7,0x0,0x8,0xd,0x5,0xf,0xb,0xb,0x5,0xa,0xe,0xf,0xe}};
int Tweak[16] =
  {0x5,0x4,0xc,0xd,0x9,0x4,0xf,0xf,0xd,0x0,0x6,0x7,0x0,0xa,0x5,0x8};
int Stt[16] =
  {0x5,0x7,0x3,0x4,0xf,0x0,0x0,0x6,0xd,0x8,0xd,0x8,0x8,0xa,0x3,0xe};

int TK[4][16];

void Initialize_key() {

  for (int i = 0; i < 16; i++) {
    TK[0][i] = Key[0][i] ^ Tweak[i];
    TK[1][i] = Key[1][i] ^ Tweak[i];
    TK[2][i] = Key[0][i] ^ Tweak[Q[i]];
    TK[3][i] = Key[1][i] ^ Tweak[Q[i]]; }

  if (dec)
    for (int j = 0; j < 4; j++)
      for (int i = 0; i < 4; i++) {
        TK[j][i] ^= (TK[j][i + 8] ^ TK[j][i + 12]);
        TK[j][i + 4] ^= TK[j][i + 12]; }
}

void Round(int r) {

  for (int i = 0; i < 4; i++) {      //MixColumn
    Stt[i] ^= (Stt[i + 8] ^ Stt[i + 12]);
    Stt[i + 4] ^= Stt[i + 12]; }

  int ind = r;
  if (dec)
    ind = 31 - r;

  Stt[4] ^= RC4[ind];                //AddConstant
  Stt[5] ^= RC3[ind];

  for (int i = 0; i < 16; i++)       //AddTweakey
    Stt[i] ^= TK[ind % 4][i];

  if (r != 31) {
    int Temp[16];
    for (int i = 0; i < 16; i++)     //Permutation
      Temp[P[i]] = Stt[i];

    for (int i = 0; i < 16; i++)     //SBox
      Stt[i] = S[Temp[i]]; }
}

int main() {

  Initialize_Tweakey();

  for (int r = 0; r < 32; r++)
    Round(r);

  return 0;
}
```

C More Implementation Details

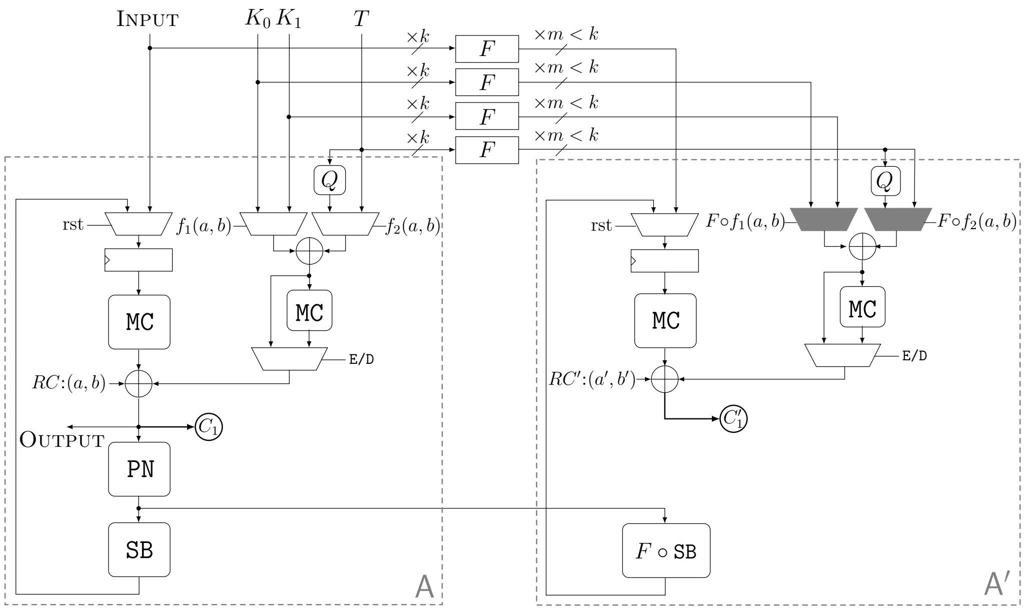


Figure 10: Round-based design architecture of CRAFT with fault detection, $m < k$ (control unit not shown, checking PN output not required).

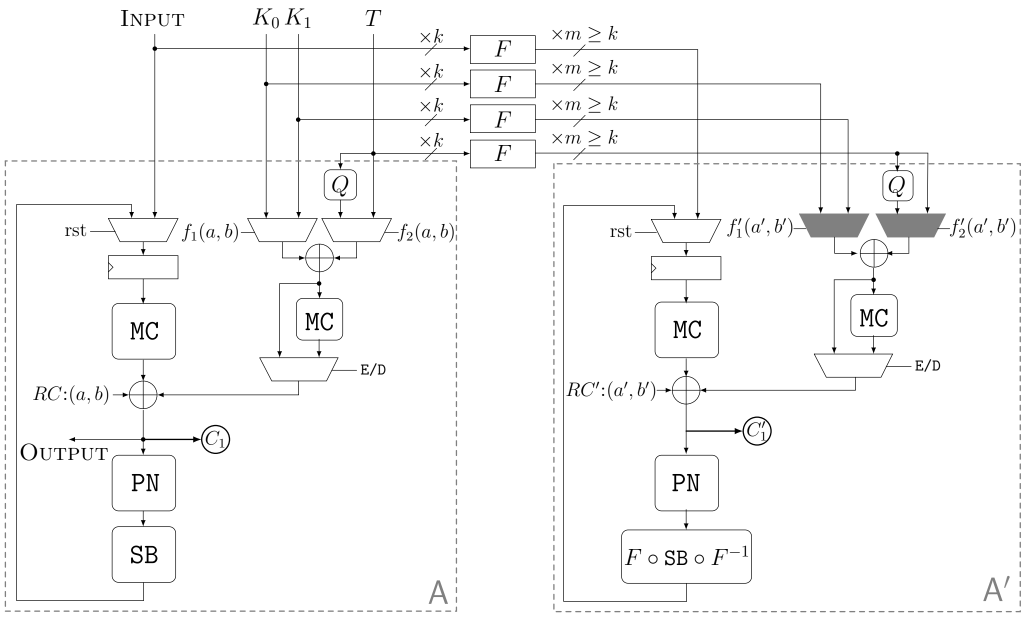


Figure 11: Round-based design architecture of CRAFT with fault detection, $m \geq k$ (control unit not shown, checking PN output not required).

Table 8: Area (GE) and Latency (ns) comparison of round-based implementations considering an $\mathcal{M}^*_{t=d-1}$-bounded multivariate adversary with an $[n, k, d]$ code, using IBM 130 nm ASIC library, partially borrowed from [1].

Algorithm	Key	clock cycles	unprotected area	unprotected latency	$[\mathbf{5, 4, 2}]$ area	$[\mathbf{5, 4, 2}]$ latency	$[\mathbf{6, 4, 2}]$ area	$[\mathbf{6, 4, 2}]$ latency	$[\mathbf{7, 4, 3}]$ area	$[\mathbf{7, 4, 3}]$ latency	$[\mathbf{8, 4, 4}]$ area	$[\mathbf{8, 4, 4}]$ latency
SKINNY Enc	128	37	1738	3.66	4236	7.84	5320	8.73	6879	8.85	8477	9.60
LED Enc	128	49	1664	9.15	4813	13.11	5729	12.46	7359	12.96	9637	15.89
MIDORI Enc	128	17	1372	7.57	3615	11.18	4358	11.42	5891	11.52	7693	14.30
PRESENT Enc	128	32	1767	2.93	4792	7.83	6015	8.53	7899	8.87	9896	9.37
GIFT Enc	128	29	1587	2.88	4420	7.49	5548	8.43	7325	8.56	9432	9.01
SIMON Enc	128	45	1629	2.86	4211	7.28	5311	7.86	6866	8.03	9277	9.97
PICCOLO Enc	128	32	1462	7.69	4196	12.81	5123	12.56	6873	12.79	9062	15.30
KATAN Enc	80	762	1080	3.87	3450	8.41	4293	9.07	5776	9.98	8092	9.38
KTANTAN Enc	80	762	601	4.23	2373	9.45	2881	9.28	3710	9.82	5073	10.94
CRAFT Enc	128	32	**949**	3.19	**2670**	7.59	**3269**	8.59	**4325**	8.22	**5864**	10.18
CRAFT Enc&Dec	128	32	**1089**	3.60	**2938**	9.08	**3583**	9.38	**4699**	9.09	**6336**	9.95
CRAFT Enc Tweak	128	32	**1193**	3.37	**3129**	8.54	**3833**	8.74	**5148**	8.23	**7507**	10.43
CRAFT Enc&Dec T.	128	32	**1339**	3.99	**3397**	9.12	**4145**	9.81	**5520**	9.98	**7982**	11.21
SKINNY Enc	192	41	2206	4.00	5272	7.69	6690	8.26	8676	8.79	10640	9.34

Table 9: Area (GE) and Latency (ns) of round-based **Threshold Implementations** of CRAFT with 3 shares considering an $\mathcal{M}^*_{t=d-1}$-bounded multivariate adversary with an $[n, k, d]$ code, using IBM 130 nm ASIC library.

Algorithm	Key	clock cycles	only TI area	only TI latency	TI+$[\mathbf{5, 4, 2}]$ area	TI+$[\mathbf{5, 4, 2}]$ latency	TI+$[\mathbf{6, 4, 2}]$ area	TI+$[\mathbf{6, 4, 2}]$ latency	TI+$[\mathbf{7, 4, 3}]$ area	TI+$[\mathbf{7, 4, 3}]$ latency	TI+$[\mathbf{8, 4, 4}]$ area	TI+$[\mathbf{8, 4, 4}]$ latency
CRAFT Enc	128	64	5106	4.05	12015	8.14	15671	8.86	18965	10.07	25482	9.83
CRAFT Enc&Dec	128	64	5303	4.33	12288	9.24	16034	9.56	19385	10.19	26043	11.11
CRAFT Enc Tweak	128	64	5412	4.17	12469	8.25	16236	8.85	19696	10.08	27121	11.97
CRAFT Enc&Dec T.	128	64	5605	4.92	12751	8.97	16598	9.94	20117	10.19	27689	11.72

D Results Using a 40nm Commercial Library

D.1 Under Univariate Adversary Model

Table 10: Area (GE) and Latency (ns) comparison of round-based implementations considering an $\mathcal{M}_{t=d-1}$-bounded univariate adversary with an $[n, k, d]$ code, using a 40 nm commercial ASIC library.

Algorithm	Key	clock cycles	unprotected area	unprotected latency	$[\mathbf{5}, \mathbf{4}, \mathbf{2}]$ area	$[\mathbf{5}, \mathbf{4}, \mathbf{2}]$ latency	$[\mathbf{6}, \mathbf{4}, \mathbf{2}]$ area	$[\mathbf{6}, \mathbf{4}, \mathbf{2}]$ latency	$[\mathbf{7}, \mathbf{4}, \mathbf{3}]$ area	$[\mathbf{7}, \mathbf{4}, \mathbf{3}]$ latency	$[\mathbf{8}, \mathbf{4}, \mathbf{4}]$ area	$[\mathbf{8}, \mathbf{4}, \mathbf{4}]$ latency
SKINNY Enc	128	37	2041	1.28	4152	2.55	5154	2.61	6457	2.89	7773	2.88
LED Enc	128	49	1940	3.98	4906	4.32	5836	4.31	7470	4.54	9658	5.48
MIDORI Enc	128	17	1616	3.31	3675	3.78	4421	3.40	5938	3.78	7724	4.29
PRESENT Enc	128	32	2050	1.04	4685	2.53	5844	2.61	7502	2.90	9270	2.96
GIFT Enc	128	29	1848	0.86	4322	2.31	5367	2.58	6904	2.81	8817	3.00
SIMON Enc	128	45	1984	0.93	4052	2.43	5076	2.55	6371	2.76	8657	2.91
PICCOLO Enc	128	32	1631	3.21	4269	4.22	5235	4.31	6945	4.84	9185	5.11
KATAN Enc	80	762	1236	1.60	3274	2.29	4047	2.36	5342	2.70	7544	2.98
KTANTAN Enc	80	762	710	1.60	2308	2.36	2792	2.46	3493	2.75	4785	2.71
CRAFT Enc	128	32	**1091**	1.66	**2660**	2.58	**3253**	2.86	**4223**	3.01	**5668**	2.89
CRAFT Enc&Dec	128	32	**1246**	2.08	**2948**	2.64	**3595**	2.73	**4627**	2.87	**6149**	2.92
CRAFT Enc Tweak	128	32	**1355**	1.81	**3167**	2.58	**3880**	2.86	**5149**	3.01	**7503**	2.89
CRAFT Enc&Dec T.	128	32	**1529**	2.02	**3454**	2.65	**4220**	2.73	**5548**	2.88	**7981**	2.92
SKINNY Enc	192	41	2592	1.33	5146	2.16	6453	2.24	8118	2.51	9735	2.62

Table 11: Area (GE) and Latency (ns) of round-based **Threshold Implementations** of CRAFT with 3 shares considering an $\mathcal{M}_{t=d-1}$-bounded univariate adversary with an $[n, k, d]$ code, using a 40 nm commercial ASIC library.

Algorithm	Key	clock cycles	only TI area	only TI latency	TI+$[\mathbf{5}, \mathbf{4}, \mathbf{2}]$ area	TI+$[\mathbf{5}, \mathbf{4}, \mathbf{2}]$ latency	TI+$[\mathbf{6}, \mathbf{4}, \mathbf{2}]$ area	TI+$[\mathbf{6}, \mathbf{4}, \mathbf{2}]$ latency	TI+$[\mathbf{7}, \mathbf{4}, \mathbf{3}]$ area	TI+$[\mathbf{7}, \mathbf{4}, \mathbf{3}]$ latency	TI+$[\mathbf{8}, \mathbf{4}, \mathbf{4}]$ area	TI+$[\mathbf{8}, \mathbf{4}, \mathbf{4}]$ latency
CRAFT Enc	128	64	5786	1.41	11974	2.48	15200	2.55	17936	2.99	24901	2.78
CRAFT Enc&Dec	128	64	6000	1.40	12273	2.40	15570	2.34	18389	2.83	25479	3.19
CRAFT Enc Tweak	128	64	6121	1.41	12482	2.48	15827	2.55	18862	2.99	26733	2.78
CRAFT Enc&Dec T.	128	64	6335	1.42	12778	2.40	16198	2.40	19308	2.82	27313	3.19

D.2 Under Multivariate Adversary Model

Table 12: Area (GE) and Latency (ns) comparison of round-based implementations considering an $\mathcal{M}^{*}_{t=d-1}$-bounded multivariate adversary with an $[n, k, d]$ code, using a 40 nm commercial ASIC library.

Algorithm	Key	clock cycles	unprotected area	unprotected latency	$[\mathbf{5}, \mathbf{4}, \mathbf{2}]$ area	$[\mathbf{5}, \mathbf{4}, \mathbf{2}]$ latency	$[\mathbf{6}, \mathbf{4}, \mathbf{2}]$ area	$[\mathbf{6}, \mathbf{4}, \mathbf{2}]$ latency	$[\mathbf{7}, \mathbf{4}, \mathbf{3}]$ area	$[\mathbf{7}, \mathbf{4}, \mathbf{3}]$ latency	$[\mathbf{8}, \mathbf{4}, \mathbf{4}]$ area	$[\mathbf{8}, \mathbf{4}, \mathbf{4}]$ latency
SKINNY Enc	128	37	2041	1.28	4835	3.50	6118	3.56	7906	3.84	9714	4.15
LED Enc	128	49	1940	3.98	5286	4.87	6314	4.99	8197	5.46	10638	6.57
MIDORI Enc	128	17	1616	3.31	4047	4.51	4896	4.86	6658	4.85	8693	5.83
PRESENT Enc	128	32	2050	1.04	5369	3.51	6807	3.56	8949	3.87	11221	4.07
GIFT Enc	128	29	1848	0.86	5006	3.44	6331	3.52	8352	3.79	10761	3.92
SIMON Enc	128	45	1984	0.93	4735	3.14	6039	3.23	7820	3.51	10600	4.30
PICCOLO Enc	128	32	1631	3.21	4648	5.52	5712	5.56	7662	6.14	10151	7.18
KATAN Enc	80	762	1236	1.60	3853	3.42	4844	3.82	6543	4.05	9158	4.51
KTANTAN Enc	80	762	710	1.60	2696	3.80	3289	3.77	4242	4.08	5794	4.71
CRAFT Enc	128	32	**1091**	1.66	**3041**	3.34	**3740**	3.57	**4961**	3.78	**6668**	4.24
CRAFT Enc&Dec	128	32	**1246**	2.08	**3327**	3.69	**4077**	3.74	**5363**	3.68	**7149**	4.50
CRAFT Enc Tweak	128	32	**1355**	1.81	**3548**	3.51	**4368**	3.69	**5887**	3.96	**8508**	4.45
CRAFT Enc&Dec T.	128	32	**1529**	2.02	**3833**	3.96	**4705**	3.62	**6283**	3.85	**8983**	4.73
SKINNY Enc	192	41	2592	1.33	5982	3.53	7658	3.58	9930	3.86	12165	4.17

Table 13: Area (GE) and Latency (ns) of round-based **Threshold Implementations** of CRAFT with 3 shares considering an $\mathcal{M}^{*}_{t=d-1}$-bounded multivariate adversary with an $[n, k, d]$ code, using a 40 nm commercial ASIC library.

Algorithm	Key	clock cycles	only TI area	only TI latency	TI+$[\mathbf{5}, \mathbf{4}, \mathbf{2}]$ area	TI+$[\mathbf{5}, \mathbf{4}, \mathbf{2}]$ latency	TI+$[\mathbf{6}, \mathbf{4}, \mathbf{2}]$ area	TI+$[\mathbf{6}, \mathbf{4}, \mathbf{2}]$ latency	TI+$[\mathbf{7}, \mathbf{4}, \mathbf{3}]$ area	TI+$[\mathbf{7}, \mathbf{4}, \mathbf{3}]$ latency	TI+$[\mathbf{8}, \mathbf{4}, \mathbf{4}]$ area	TI+$[\mathbf{8}, \mathbf{4}, \mathbf{4}]$ latency
CRAFT Enc	128	64	5786	1.41	13580	3.68	17354	3.51	21226	3.83	29339	4.44
CRAFT Enc&Dec	128	64	6000	1.40	13888	3.96	17721	3.51	21697	4.19	29921	4.79
CRAFT Enc Tweak	128	64	6121	1.41	14088	3.82	17982	3.60	22150	3.83	31178	4.64
CRAFT Enc&Dec T.	128	64	6335	1.42	14393	4.11	18352	3.72	22620	3.89	31761	5.10