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

Christof Beierle¹, Gregor Leander², Amir Moradi² and Shahram Rasoolzadeh²

¹ SnT, University of Luxembourg, Luxembourg
beierle.christof@gmail.com

² Ruhr University Bochum, Horst Görtz Institute for IT Security, 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 to¹ integrate such

¹It is forced if particular certification is needed, e.g. common criteria evaluation.
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 cipher⁴. It indeed competes with a bit-serial implementation of SIMON with 958 GE requiring $64 \times 44$ 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 countering 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 coverage⁵ 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

---

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

⁵Fault coverage is calculated by $\sum_{d} / \sum_{v}$ with $\sum_{d}$ the number of detectable and $\sum_{v}$ the number of possible faults.
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 provide 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 $C$ 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 $x \in \mathbb{F}_2^l$ into the corresponding codeword $c \in \mathbb{F}_2^k$ with $c = x \cdot G$. The minimum distance $d$ is defined as

$$\min \{|c_1 + c_2| \mid c_1, c_2 \in C, c_1 \neq c_2\},$$

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 $C$ 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 $C$ 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].
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 $\otimes$ and $\oplus$ 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 $c'' \neq c'$ 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\textsuperscript{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 $\text{INPUT}'$ in Figure 2(a)).

We show here that in the construction of Figure 2(a) if $m < k$, for any function $F : \mathbb{F}_2^k \rightarrow \mathbb{F}_2^m$, 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))

\textsuperscript{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 $k$, denoted by $E_k$, 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 $E_k$, there exists a block cipher $E'_k$ such that $E'_k \circ F = F' \circ E_k$. Thus, for every two plaintexts $p_1$ and $p_2$ for which $F(p_1) = F(p_2)$, one obtains $F' \circ E_k(p_1) = F' \circ E_k(p_2)$. 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 $F^{-1}(x), x \in \mathbb{Z}_2^m$ is a set containing $2^{k-m}$ elements (the same holds for $F'^{-1}$). Further, every instance $E_k$ operates as a permutation over these preimages, i.e.,

$$E_k : F^{-1}(x) \mapsto F'^{-1}(y), \text{ if } E'_k : x \mapsto y.$$

Thus, under a chosen-plaintext attack, the underlying cipher $E_k$ can easily be distinguished from a random permutation. The adversary chooses plaintexts $p_1$ and $p_2$ with $F(p_1) = F(p_2)$. If $c_i = E_k(p_i)$, the property $F'(c_1) = F'(c_2)$ holds with a probability of 1. Note that this property should hold only with probability $2^{-m}$ for a uniformly chosen random permutation $E_k : p_i \mapsto c_i$.

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 $4 \times 4$ 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 $4 \times 4$ 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, $I_{i,j} = I_{4i+j}$. 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 these 64-bit tweakeys is also considered as a $4 \times 4$ 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 ($R_i, 0 \leq i \leq 30$) and appends one more linear round $R_{31}$ to compute the ciphertext. Figure 3 depicts the structure of CRAFT. Each round function $R_i$ applies the following five involutory round operations: SubBox, MixColumn, PermuteNibbles, AddConstant, and AddTweakey, while $R_{31}$ only applies the MixColumn, AddConstant, and AddTweakey 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

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>S(x)</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>3</td>
<td>e</td>
<td>b</td>
<td>f</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

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 $j \in \{0, \ldots, 3\}$,

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

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

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

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

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

AddTweakey ($ATK_i$): 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 $Q = [12, 10, 15, 5, 14, 8, 9, 2, 11, 3, 7, 4, 6, 0, 1, 13]$ on the nibbles of the tweak $T$ where for all $0 \leq i \leq 15$, $T_i$ is replaced by $T_{Q(i)}$. Then in each round $i$, without any key update, the tweakey $TK_i \mod 4$ is XOR-ed to the cipher state.

Round Function: To conclude, using the above explained operations, the round functions $R_i$, $i \in \{0, \ldots, 30\}$, are defined as $R_i = SB \circ PN \circ ATK_i \circ ARC_i \circ MC$ and the last round $R'_{31}$ as $R'_{31} = ATK_{31} \circ ARC_{31} \circ MC$.

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 $R'_{31}$, 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 $(RC_0, \ldots, RC_{31})$ is the same as the CRAFT encryption with tweakeys $(TK'_0, TK'_1, TK'_2, TK'_3)$ and round constants $(RC_{31}, \ldots, RC_0)$, where $TK'_i = MC(TK_i)$.

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

\[
\begin{bmatrix}
0 \\
x \\
0 \\
0
\end{bmatrix}
\]

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.

<table>
<thead>
<tr>
<th>Algorithm 4.1: Encryption</th>
<th>Algorithm 4.2: Decryption</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong>: X: plaintext</td>
<td><strong>Input</strong>: X: ciphertext</td>
</tr>
<tr>
<td>K₀</td>
<td></td>
</tr>
<tr>
<td>T: tweak</td>
<td>T: tweak</td>
</tr>
<tr>
<td><strong>Output</strong>: Y: ciphertext</td>
<td><strong>Output</strong>: Y: plaintext</td>
</tr>
</tbody>
</table>

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)
  Y₄₅ ← Y₄₅ ⊕ RCᵢ
  Y ← Y ⊕ TKᵢ mod 4
  if i ≠ 31 then
    Y ← PN(Y)
  end
end

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)
  Y₄₅ ← Y₄₅ ⊕ RCᵢ
  Y ← Y ⊕ TKᵢ mod 4
  if i ≠ 0 then
    Y ← PN(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 \( T : \mathbb{F}_2^q \rightarrow \mathbb{F}_2^q \) which maps the input \( x \) to a \( q \)-bit output \( y : \langle y¹, \ldots, y^q \rangle \). The function \( T(x) = y \) is physically realized by \( q \) component circuits each of which realizing a coordinate function \( T^i : \mathbb{F}_2^q \rightarrow \mathbb{F}_2 \) 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,

\[
\forall i, j; \ i \neq j \quad \mathcal{G}^i \cap \mathcal{G}^j = \emptyset,
\]

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 \( \text{one} \) 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:

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

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 \( P_1 \) 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 \( S'_1 = F_4 \circ S \), 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 \( S, S_1, \ldots, S_4 \), we need to know the size of the implementation of \( S, S_1, S'_1 \) 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 12870 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 130nm 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 \( S, S_1, S'_1 \) 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 130nm ASIC library.

<table>
<thead>
<tr>
<th>Class</th>
<th>(u,l)</th>
<th>(n,m)</th>
<th>Sbox</th>
<th>Size [GE]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(S, S₁)</td>
</tr>
<tr>
<td>266</td>
<td>(4,4)</td>
<td>(32,32)</td>
<td>9BDFAE678041C253</td>
<td>11</td>
</tr>
<tr>
<td>262</td>
<td>(8,4)</td>
<td>(64,32)</td>
<td>0189A5E237DF666C</td>
<td>12.25</td>
</tr>
<tr>
<td>45</td>
<td>(6,6)</td>
<td>(46,78)</td>
<td>016E42795ACBD35</td>
<td>12</td>
</tr>
<tr>
<td>51</td>
<td>(6,6)</td>
<td>(46,78)</td>
<td>87A39F816420DCE5</td>
<td>12.5</td>
</tr>
<tr>
<td>76</td>
<td>(6,6)</td>
<td>(46,78)</td>
<td>E6AFDS178C2B9403</td>
<td>12.75</td>
</tr>
<tr>
<td>208</td>
<td>(6,6)</td>
<td>(46,78)</td>
<td>413205B7CEA68F9D</td>
<td>12.25</td>
</tr>
<tr>
<td>24</td>
<td>(8,6)</td>
<td>(64,32)</td>
<td>6C284E0F39BA1D57</td>
<td>13</td>
</tr>
<tr>
<td>32</td>
<td>(8,6)</td>
<td>(64,32)</td>
<td>816E4F2798ACBD35</td>
<td>14</td>
</tr>
<tr>
<td>46</td>
<td>(8,6)</td>
<td>(64,32)</td>
<td>87A395816420DCEF</td>
<td>12.5</td>
</tr>
<tr>
<td>57</td>
<td>(8,6)</td>
<td>(64,32)</td>
<td>DF5754263B94E0C1</td>
<td>11.25</td>
</tr>
<tr>
<td>101</td>
<td>(8,6)</td>
<td>(64,32)</td>
<td>2103D6CAE8E75B9</td>
<td>14</td>
</tr>
<tr>
<td>102</td>
<td>(8,6)</td>
<td>(64,32)</td>
<td>290D648C81A573FE</td>
<td>13</td>
</tr>
<tr>
<td>137</td>
<td>(10,6)</td>
<td>(95,78)</td>
<td>01C6F64E89BAB2375</td>
<td>12</td>
</tr>
<tr>
<td>141</td>
<td>(10,6)</td>
<td>(95,78)</td>
<td>98DCF78510B83E4</td>
<td>12.75</td>
</tr>
<tr>
<td>149</td>
<td>(10,6)</td>
<td>(95,78)</td>
<td>465ED21FCDAA937</td>
<td>12</td>
</tr>
<tr>
<td>217</td>
<td>(10,6)</td>
<td>(95,78)</td>
<td>6523410BEDF7C96A</td>
<td>11</td>
</tr>
<tr>
<td>216</td>
<td>(12,6)</td>
<td>(155,78)</td>
<td>2301AB7CF4989EE</td>
<td>10.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>E6A4371555B9A92F6DC</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>E6C43715B9A02F0DC</td>
<td>11.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6798AB012345678D</td>
<td>11.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>54A6E617306B29F8DC</td>
<td>12.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2108F0987A5E504</td>
<td>12.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EAC86754F5B412D07</td>
<td>13</td>
</tr>
</tbody>
</table>

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 20160 bijective \( 4 \times 4 \) 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 $P_r$ be a permutation over the rows of the state. The encryption with round operations $SB$, $MC$ and $PN$ with matrix $M$ and permutation $P$ 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 $M' = P_{r^{-1}} \circ M \circ P_{r}$ and $PN'$ applies $P' = P^{-1}_{c} \circ P \circ P_{c}$.

The 30 candidates for the matrix $M$ are given below.

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

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

$M_{10} : \begin{bmatrix} 1010 \\ 0111 \\ 0011 \\ 0001 \end{bmatrix}$  
$M_{11} : \begin{bmatrix} 0101 \\ 1011 \\ 0011 \\ 0001 \end{bmatrix}$  
$M_{12} : \begin{bmatrix} 0101 \\ 1011 \\ 0011 \\ 0001 \end{bmatrix}$  
$M_{13} : \begin{bmatrix} 1001 \\ 0100 \\ 0010 \\ 0001 \end{bmatrix}$  
$M_{14} : \begin{bmatrix} 1001 \\ 0011 \\ 0001 \\ 0001 \end{bmatrix}$

$M_{15} : \begin{bmatrix} 1011 \\ 0111 \\ 0011 \\ 0001 \end{bmatrix}$  
$M_{16} : \begin{bmatrix} 0111 \\ 1011 \\ 0011 \\ 0001 \end{bmatrix}$  
$M_{17} : \begin{bmatrix} 0111 \\ 1011 \\ 0011 \\ 0001 \end{bmatrix}$  
$M_{18} : \begin{bmatrix} 1001 \\ 0100 \\ 0010 \\ 0001 \end{bmatrix}$  
$M_{19} : \begin{bmatrix} 1001 \\ 0011 \\ 0001 \\ 0001 \end{bmatrix}$

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

$M_{25} : \begin{bmatrix} 1111 \\ 1111 \\ 1110 \\ 1110 \end{bmatrix}$  
$M_{26} : \begin{bmatrix} 1010 \\ 1111 \\ 1110 \\ 1110 \end{bmatrix}$  
$M_{27} : \begin{bmatrix} 1101 \\ 1111 \\ 1110 \\ 1110 \end{bmatrix}$  
$M_{28} : \begin{bmatrix} 1101 \\ 1111 \\ 1110 \\ 1110 \end{bmatrix}$  
$M_{29} : \begin{bmatrix} 1101 \\ 1111 \\ 1110 \\ 1110 \end{bmatrix}$

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 $P_c$ be a permutation over the columns of the state. The encryption with round operations $SB$, $MC$ and $PN$ with permutation $P$ 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 $P' = P^{-1}_{c} \circ P \circ P_{c}$.

Therefore, we can reduce the search space of all involutory permutations $P$ by the above equivalence. 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 $P$, the matrices $M_0, \ldots, M_{11}$ 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 $\mathcal{MC}$

| $M_i$ | 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      | 4  | 5  | 7  | 8  |
| #mux  | 4  | 2  | 3      | 2      | 3  | 4      | 3  | 4  | 4  |
| $r_1$ | 8  | 7  | 7      | 6      | 6  | 5      | 6  | 5  | 4  | 4  | 4  | 4  | 4       |
| $r_2$ | 10 | 9  | 11     | 16     | 9  | 8      | 16 | 8  | 8  | 11 | 8  | 7  |

while in decryption $\mathcal{MC}(TK_i)$ 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 $\mathcal{MC}$ 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 $\Delta = (x,x,\ldots,x)$ and $x \in \mathbb{F}_2^4$. Then we have $Q(\Delta) = \Delta$ which cause the relations below.

\[

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_2' + 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.
\]

This means encryption under two different twekey 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 $T, T + \Delta_1, \ldots, T + \Delta_{15}$ where $\Delta_2 = (x, x, \ldots, x)$ that we show the corresponding ciphertexts by $C_0, C_1, \ldots, C_{15}$. Then, by setting one of the key nibbles to constant value of zero, for each of $2^{124}$ possible key candidates $(K_0^*, K_1^*)$, he computes $C^*$, the encryption of $P$ using $K_0^*, K_1^*$ and $T$. If $C^*$ is equal to $C_2$, then $(K_0^* + \Delta_2, K_1^* + \Delta_2)$ 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 twekey 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 twekeys 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 $T$. 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 $T$. 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 computations 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, $2^{84}$ 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. RT$_i$ refers to the characteristic starting with round $R_{i+4}$.

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

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 characteristic, 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 $2^{-62.61}$:

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

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

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

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 $2^{-55.14}$:

$$(0, 0, 0, 0, a, 0, 0, 0, 0, 0, 0, 0, 0, 0, a, 0, a, 0, 0, 0)$$

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

$$(0, a, 0, a, 0, a, 0, 0, 0, 0, 0, 0, 0, 0, 0, a, 0, 0, 0, 0, 0)$$

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

$$(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, a, 0, a, 0, 0, 0, 0)$$

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

$$(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, a, a, 0, a, 0, 0, 0, 0)$$
where for all of them $\Delta T$ is equal to $(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$.

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 $(\Gamma, \Delta)$ 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 $\Gamma$ and $\Delta$, 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)\footnote{Note that our approach only takes the properties of the linear layer into account and is independent of the choice of the Sbox.} 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 $\gamma$ and $\delta$ can take any non-zero difference in $F_4^3$. 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.

\[
\begin{align*}
(0, 0, 0, \gamma, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) & \rightarrow (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
(0, 0, 0, \gamma, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) & \rightarrow (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
(0, 0, 0, \gamma, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) & \rightarrow (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
(0, 0, 0, \gamma, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) & \rightarrow (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
(0, 0, \gamma, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) & \rightarrow (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
(0, 0, \gamma, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) & \rightarrow (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
(0, 0, \gamma, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) & \rightarrow (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
(0, 0, \gamma, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) & \rightarrow (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
(0, \gamma, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) & \rightarrow (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
(0, \gamma, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) & \rightarrow (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
(\gamma, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) & \rightarrow (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
(\gamma, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) & \rightarrow (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
(\gamma, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) & \rightarrow (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
\end{align*}\]
where $\gamma$ and $\delta$ can take any non-zero mask in $F_2^4$.

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 \(\mathbb{F}_4^2\). 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 \(\mathbb{F}_4^2\) 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 \(g + g \circ E_k\) is a
constant function. Such a $g$ is called an invariant for $E_k$. 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 $R_i$ and $R_j$ be two rounds in which the same round tweakey is added and let $d_{i,j} := \text{PN} \circ \text{MC}(R_i) \oplus \text{PN} \circ \text{MC}(R_j)$. If the smallest ($\text{PN} \circ \text{MC}$)-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 $\text{PN} \circ \text{MC} \circ \text{ARC}_i \circ \text{ATK}_i$ and $\text{PN} \circ \text{MC} \circ \text{ARC}_j \circ \text{ATK}_j$ 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 $\text{PN} \circ \text{MC}$-invariant subspace that contains $\{d_{0,4}, d_{1,5}, d_{2,6}, d_{4,7}\}$ is $F_{64}^4$, i.e., it has dimension 64. Therefore, we already get a sound security argument on the resistance against invariant attacks after 8 rounds.

Figure 6: A 15-round partial matching meet-in-the-middle 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 \( MC(TK_i) \) 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 \( MC(TK_i) \) 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, \( \text{D00E}, \text{D01} \) generates 0001,001 for encryption (\( E = 1 \)) and 1000,101 for decryption (\( E = 0 \)). 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 7: A 13-round integral distinguisher for CRAFT.](image-url)
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 [8], 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\textsuperscript{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 tweak key 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

\textsuperscript{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 130nm ASIC library, partially borrowed from [1].

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Key</th>
<th>clock cycles</th>
<th>unprotected cycles</th>
<th>area latency [5.4.2]</th>
<th>area latency [6.4.2]</th>
<th>area latency [7.4.3]</th>
<th>area latency [8.4.4]</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKINNY Enc</td>
<td>128</td>
<td>37</td>
<td>1738</td>
<td>3.66</td>
<td>3640</td>
<td>4494</td>
<td>56.36</td>
</tr>
<tr>
<td>LED Enc</td>
<td>128</td>
<td>49</td>
<td>1664</td>
<td>9.15</td>
<td>4499</td>
<td>5264</td>
<td>6699</td>
</tr>
<tr>
<td>MIDORI Enc</td>
<td>128</td>
<td>17</td>
<td>1372</td>
<td>7.57</td>
<td>3282</td>
<td>3942</td>
<td>5262</td>
</tr>
<tr>
<td>PRESENT Enc</td>
<td>128</td>
<td>32</td>
<td>1767</td>
<td>2.93</td>
<td>4211</td>
<td>5177</td>
<td>6639</td>
</tr>
<tr>
<td>GIFT Enc</td>
<td>128</td>
<td>29</td>
<td>1587</td>
<td>2.88</td>
<td>3824</td>
<td>4722</td>
<td>6082</td>
</tr>
<tr>
<td>SIMON Enc</td>
<td>128</td>
<td>45</td>
<td>1629</td>
<td>2.86</td>
<td>4614</td>
<td>4487</td>
<td>5621</td>
</tr>
<tr>
<td>PICOCO Enc</td>
<td>128</td>
<td>32</td>
<td>1462</td>
<td>7.69</td>
<td>3870</td>
<td>4763</td>
<td>6241</td>
</tr>
<tr>
<td>KATAN Enc</td>
<td>80</td>
<td>762</td>
<td>1080</td>
<td>3.87</td>
<td>2946</td>
<td>3610</td>
<td>4746</td>
</tr>
<tr>
<td>KTANTAN Enc</td>
<td>80</td>
<td>762</td>
<td>601</td>
<td>4.23</td>
<td>2039</td>
<td>2457</td>
<td>3069</td>
</tr>
<tr>
<td>CRAFT Enc</td>
<td>128</td>
<td>32</td>
<td>949</td>
<td>3.19</td>
<td>2342</td>
<td>2857</td>
<td>3698</td>
</tr>
<tr>
<td>CRAFT Enck/Dec</td>
<td>128</td>
<td>32</td>
<td>1089</td>
<td>3.60</td>
<td>2609</td>
<td>3169</td>
<td>4069</td>
</tr>
<tr>
<td>CRAFT Enc Tweak</td>
<td>128</td>
<td>32</td>
<td>1193</td>
<td>3.37</td>
<td>2801</td>
<td>3420</td>
<td>4518</td>
</tr>
<tr>
<td>CRAFT Enck/Dec T</td>
<td>128</td>
<td>32</td>
<td>1339</td>
<td>3.99</td>
<td>3066</td>
<td>3731</td>
<td>4891</td>
</tr>
<tr>
<td>SKINNY Enc</td>
<td>192</td>
<td>41</td>
<td>2206</td>
<td>4.00</td>
<td>4540</td>
<td>5656</td>
<td>7119</td>
</tr>
</tbody>
</table>

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 130nm 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 40nm standard cell library. 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.

\footnote{Due to an NDA, the full name of the used library is omitted.}
• 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 $\mathcal{MC}$ of CRAFT has been chosen to 1) let $\mathcal{MC}$ operate solely on the redundant part of information for $m < k$ (see Figure 10), and 2) avoid any necessary extra check point at $\mathcal{MC}$ 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 40nm 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 $M_{t = d-1}$-bounded univariate adversary with an $[n,k,d]$ code, using the IBM 130nm ASIC library.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CRAFT Enc</td>
<td>128</td>
<td>64</td>
<td>5.106</td>
<td>4.05</td>
<td>5.63</td>
<td>13.85</td>
<td>6.75</td>
</tr>
<tr>
<td>CRAFT Enc&amp;Dec</td>
<td>128</td>
<td>64</td>
<td>5.983</td>
<td>4.33</td>
<td>6.46</td>
<td>14.19</td>
<td>6.28</td>
</tr>
<tr>
<td>CRAFT Enc Tweak</td>
<td>128</td>
<td>64</td>
<td>5.412</td>
<td>4.17</td>
<td>5.63</td>
<td>14.41</td>
<td>5.80</td>
</tr>
<tr>
<td>CRAFT Enc&amp;Dec T.</td>
<td>128</td>
<td>64</td>
<td>5.685</td>
<td>4.92</td>
<td>6.37</td>
<td>14.76</td>
<td>6.27</td>
</tr>
</tbody>
</table>

CRAFT only-encryption (without tweak) protected against a multivariate adversary $M_{t = 1}$ 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 $M_t$ 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 40nm 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


## A CRAFT Test Vectors

| \(K_{11}\) | 0000000000000000 | 0123456789ABCDEF | 023456789ABCDEF | 0123456789ABCDEF |
| \(K_{11}\) | 0000000000000000 | 0123456789ABCDEF | 0123456789ABCDEF | 023456789ABCDEF |
| \(T\) | 0000000000000000 | 0123456789ABCDEF | 0123456789ABCDEF | 023456789ABCDEF |
| \(T_{K0}\) | 0000000000000000 | 0123456789ABCDEF | 0123456789ABCDEF | 73B3EC59346EE4 |
| \(T_{K1}\) | 0000000000000000 | 0123456789ABCDEF | 0123456789ABCDEF | 73B3EC59346EE4 |
| \(T_{K2}\) | 0000000000000000 | 0123456789ABCDEF | 0123456789ABCDEF | 73B3EC59346EE4 |
| \(T_{K3}\) | 0000000000000000 | 0123456789ABCDEF | 0123456789ABCDEF | 73B3EC59346EE4 |
| \(P\) | 0000000000000000 | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_1\) | CCCCCCCCAACCCCC | CCCCCCCCAACCCCC | CCCCCCCCAACCCCC | CCCCCCCCAACCCCC |
| \(R_2\) | 0000110000000000 | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_3\) | CAC8400000000000 | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_4\) | DF8C200000000000 | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_5\) | 6F2C920000000000 | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_6\) | 78EBADFE88BBB30 | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_7\) | C5385454DF684BD | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_8\) | 2856E6C850000000 | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_9\) | 018B0BF0F0343AE | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{10}\) | 4E313C63B8F85B8 | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{11}\) | BF855162A80F359 | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{12}\) | 9C3B81D6CE56FB39 | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{13}\) | 9653840F42689B1E | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{14}\) | A9549DFE40B463EE | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{15}\) | 4F345CEC62A1C48 | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{16}\) | 8A0EDF0110EBF0BF | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{17}\) | 956564A6F486EB3D | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{18}\) | 2F48EF6D675EB959 | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{19}\) | 9497F2B909DACCEA | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{20}\) | 1F0429C10EAA113B | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{21}\) | 51A324C16E16458 | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{22}\) | BE545AFA7ADEC67B | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{23}\) | 50F7217493933ECC | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{24}\) | 034039E21587C4A | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{25}\) | E014DB8DC4B20CD | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{26}\) | 20C0E2552787E7A | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{27}\) | 1747B75897ED20C | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{28}\) | 042D795F9913A2E | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{29}\) | 431DA9997DA47A11 | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(R_{30}\) | A71A3271874E53F | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
| \(C\) | 4A36E78883063100 | 0123456789ABCDEF | 0123456789ABCDEF | 0123456789ABCDEF |
B  CRAFT C++ Code

```cpp
const int S[16] = {0xc, 0xa, 0xd, 0x3, 0xce, 0xb, 0xf, 0x7, 0x8, 0x9, 0x1, 0x5, 0x0, 0x2, 0x4, 0x6};
const int P[16] = {0xf, 0xce, 0xe, 0xa, 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, 0x6, 0x5, 0xe, 0x7, 0x3, 0x1, 0x4,
0x2, 0x5, 0x6, 0x7, 0x3, 0x1, 0x4, 0x4, 0x5, 0x6, 0x7, 0x3, 0x1, 0x4, 0x2, 0x5} ;
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, 0x4, 0x5, 0x6, 0x7, 0x3, 0x1, 0x4, 0x2, 0x5};
const int RC4[32] = {0x1, 0x5, 0x4, 0x2, 0x9, 0x6, 0xb, 0x3, 0xa, 0x0, 0xe, 0xf, 0x7, 0x3, 0x1,
0x2, 0x5, 0x6, 0x7, 0x3, 0x1, 0x4, 0x4, 0x5, 0x6, 0x7, 0x3, 0x1, 0x4, 0x2, 0x5};
const bool dec = 0; // encryption: 0, decryption: 1
int Key[2][16] = {
{0x2, 0x7, 0xa, 0xe, 0x6, 0x7, 0x8, 0x1, 0xa, 0x0, 0xc, 0x3, 0x0, 0x4, 0x6, 0xb, 0xc},
{0xe9, 0x1, 0x6, 0x7, 0x0, 0x8, 0x0, 0x6, 0x5, 0x0, 0xe, 0x3, 0x8, 0x0, 0x9, 0x0, 0xe} } ;
int Tweak[16] = {0x5, 0x4, 0xc, 0xd, 0x9, 0x4, 0x0, 0xf, 0xe, 0xe, 0x0, 0bx, 0xe, 0xe, 0xc, 0xe, 0x0} ;
int Stt[16] = {0x5, 0x7, 0x3, 0x4, 0x0, 0x0, 0x6, 0x0, 0x8, 0x0, 0x8, 0xe, 0x7, 0x3, 0x1, 0x0, 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]]; 
}
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];
Stt[0] = 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 $M^*_t = d-1$-bounded multivariate adversary with an $[n,k,d]$ code, using IBM 130nm ASIC library, partially borrowed from [1].

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Key</th>
<th>cycles</th>
<th>clock</th>
<th>area</th>
<th>latency</th>
<th>area</th>
<th>latency</th>
<th>area</th>
<th>latency</th>
<th>area</th>
<th>latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKINNY Enc</td>
<td></td>
<td>128</td>
<td>37</td>
<td>1738</td>
<td>3.66</td>
<td>4236</td>
<td>7.84</td>
<td>5320</td>
<td>8.73</td>
<td>6879</td>
<td>8.85</td>
</tr>
<tr>
<td>LED Enc</td>
<td></td>
<td>128</td>
<td>49</td>
<td>1604</td>
<td>9.15</td>
<td>4813</td>
<td>13.11</td>
<td>5729</td>
<td>12.46</td>
<td>7359</td>
<td>12.96</td>
</tr>
<tr>
<td>MIBRI Enc</td>
<td></td>
<td>128</td>
<td>17</td>
<td>1372</td>
<td>7.57</td>
<td>8615</td>
<td>11.18</td>
<td>4358</td>
<td>11.42</td>
<td>5891</td>
<td>11.52</td>
</tr>
<tr>
<td>PRESENT Enc</td>
<td></td>
<td>128</td>
<td>32</td>
<td>1707</td>
<td>2.93</td>
<td>4792</td>
<td>7.83</td>
<td>6015</td>
<td>8.53</td>
<td>7899</td>
<td>8.87</td>
</tr>
<tr>
<td>GIFT Enc</td>
<td></td>
<td>128</td>
<td>29</td>
<td>1587</td>
<td>2.88</td>
<td>4420</td>
<td>7.49</td>
<td>5548</td>
<td>8.43</td>
<td>7325</td>
<td>8.56</td>
</tr>
<tr>
<td>SIMON Enc</td>
<td></td>
<td>128</td>
<td>45</td>
<td>1629</td>
<td>2.86</td>
<td>4211</td>
<td>7.28</td>
<td>5311</td>
<td>8.66</td>
<td>6866</td>
<td>8.03</td>
</tr>
<tr>
<td>PICCOLO Enc</td>
<td></td>
<td>128</td>
<td>32</td>
<td>1402</td>
<td>7.69</td>
<td>4196</td>
<td>12.81</td>
<td>5123</td>
<td>12.56</td>
<td>6873</td>
<td>12.79</td>
</tr>
<tr>
<td>KATAN Enc</td>
<td></td>
<td>80</td>
<td>762</td>
<td>1080</td>
<td>3.87</td>
<td>3450</td>
<td>8.41</td>
<td>4293</td>
<td>9.07</td>
<td>5776</td>
<td>9.98</td>
</tr>
<tr>
<td>KTANTAN Enc</td>
<td></td>
<td>80</td>
<td>762</td>
<td>601</td>
<td>4.23</td>
<td>2373</td>
<td>9.45</td>
<td>2881</td>
<td>9.28</td>
<td>5773</td>
<td>9.82</td>
</tr>
<tr>
<td>CRAFT Enc</td>
<td></td>
<td>128</td>
<td>32</td>
<td>949</td>
<td>3.19</td>
<td>2670</td>
<td>7.59</td>
<td>3269</td>
<td>8.59</td>
<td>4325</td>
<td>8.22</td>
</tr>
<tr>
<td>CRAFT Enc&amp;Dec</td>
<td></td>
<td>128</td>
<td>32</td>
<td>1089</td>
<td>3.60</td>
<td>2938</td>
<td>9.08</td>
<td>3583</td>
<td>9.38</td>
<td>4099</td>
<td>9.09</td>
</tr>
<tr>
<td>CRAFT Enc Tweak</td>
<td></td>
<td>128</td>
<td>32</td>
<td>1193</td>
<td>3.37</td>
<td>3129</td>
<td>8.54</td>
<td>3833</td>
<td>8.74</td>
<td>5148</td>
<td>8.23</td>
</tr>
<tr>
<td>CRAFT Enc</td>
<td></td>
<td>192</td>
<td>41</td>
<td>2206</td>
<td>4.00</td>
<td>5272</td>
<td>7.69</td>
<td>6690</td>
<td>8.26</td>
<td>8676</td>
<td>8.79</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Key</th>
<th>clock</th>
<th>only TI</th>
<th>TI+ [5.4.2]</th>
<th>TI+ [6.4.2]</th>
<th>TI+ [7.4.3]</th>
<th>TI+ [8.4.4]</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRAFT Enc</td>
<td></td>
<td>128</td>
<td>64</td>
<td>5106</td>
<td>4.05</td>
<td>12015</td>
<td>8.14</td>
</tr>
<tr>
<td>CRAFT Enc&amp;Dec</td>
<td></td>
<td>128</td>
<td>64</td>
<td>5383</td>
<td>4.33</td>
<td>12288</td>
<td>9.24</td>
</tr>
<tr>
<td>CRAFT Enc Tweak</td>
<td></td>
<td>128</td>
<td>64</td>
<td>5412</td>
<td>4.17</td>
<td>12469</td>
<td>8.25</td>
</tr>
<tr>
<td>CRAFT Enc&amp;Dec T.</td>
<td></td>
<td>128</td>
<td>64</td>
<td>5665</td>
<td>4.92</td>
<td>12751</td>
<td>8.97</td>
</tr>
</tbody>
</table>
## 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 $M_{t=d-1}$-bounded univariate adversary with an $[n,k,d]$ code, using a 40nm commercial ASIC library.

<table>
<thead>
<tr>
<th>Algorithm Key</th>
<th>Algorithm</th>
<th>clock cycles</th>
<th>unprotected area (GE)</th>
<th>[5, 4, 2] area (GE)</th>
<th>[6, 4, 2] area (GE)</th>
<th>[7, 4, 3] area (GE)</th>
<th>[8, 4, 4] area (GE)</th>
<th>area latency (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKINNY Enc 128</td>
<td>37</td>
<td>2041</td>
<td>1.28</td>
<td>4152</td>
<td>5154</td>
<td>6457</td>
<td>7773</td>
<td>2.89</td>
</tr>
<tr>
<td>LED Enc 128</td>
<td>49</td>
<td>1940</td>
<td>3.98</td>
<td>4906</td>
<td>5836</td>
<td>7470</td>
<td>9658</td>
<td>5.48</td>
</tr>
<tr>
<td>MIDORI Enc 128</td>
<td>17</td>
<td>1616</td>
<td>3.31</td>
<td>3675</td>
<td>4421</td>
<td>5938</td>
<td>7724</td>
<td>4.29</td>
</tr>
<tr>
<td>PRESENT Enc</td>
<td>32</td>
<td>2050</td>
<td>1.04</td>
<td>4685</td>
<td>5844</td>
<td>7502</td>
<td>9270</td>
<td>2.96</td>
</tr>
<tr>
<td>GIFT Enc 128</td>
<td>29</td>
<td>1848</td>
<td>0.86</td>
<td>4322</td>
<td>5367</td>
<td>6904</td>
<td>8817</td>
<td>3.00</td>
</tr>
<tr>
<td>SIMON Enc 128</td>
<td>45</td>
<td>1984</td>
<td>0.93</td>
<td>4052</td>
<td>5076</td>
<td>6471</td>
<td>8657</td>
<td>2.91</td>
</tr>
<tr>
<td>PRESENT Enc</td>
<td>32</td>
<td>1631</td>
<td>3.21</td>
<td>4269</td>
<td>5235</td>
<td>6945</td>
<td>9185</td>
<td>5.11</td>
</tr>
<tr>
<td>KATAN Enc</td>
<td>762</td>
<td>1236</td>
<td>1.60</td>
<td>3274</td>
<td>4047</td>
<td>5442</td>
<td>7544</td>
<td>2.98</td>
</tr>
<tr>
<td>KTANTAN Enc</td>
<td>762</td>
<td>710</td>
<td>1.60</td>
<td>2308</td>
<td>2792</td>
<td>3493</td>
<td>4785</td>
<td>2.71</td>
</tr>
<tr>
<td>CRAFT Enc 128</td>
<td>32</td>
<td>1091</td>
<td>1.66</td>
<td>2660</td>
<td>3253</td>
<td>4223</td>
<td>5668</td>
<td>2.89</td>
</tr>
<tr>
<td>CRAFT Enc&amp;Dec</td>
<td>32</td>
<td>1246</td>
<td>2.08</td>
<td>2948</td>
<td>3595</td>
<td>4627</td>
<td>6149</td>
<td>2.92</td>
</tr>
<tr>
<td>CRAFT Enc Tweak</td>
<td>32</td>
<td>1355</td>
<td>1.81</td>
<td>3167</td>
<td>3880</td>
<td>5149</td>
<td>7503</td>
<td>2.89</td>
</tr>
<tr>
<td>CRAFT Enc&amp;Dec T.</td>
<td>32</td>
<td>1529</td>
<td>2.02</td>
<td>3454</td>
<td>4220</td>
<td>5548</td>
<td>7981</td>
<td>2.92</td>
</tr>
<tr>
<td>SKINNY Enc 192</td>
<td>41</td>
<td>2592</td>
<td>1.33</td>
<td>5146</td>
<td>6453</td>
<td>8118</td>
<td>9735</td>
<td>2.62</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Algorithm Key</th>
<th>Algorithm</th>
<th>clock cycles</th>
<th>only TI [5, 4, 2] area (GE)</th>
<th>TI+ [6, 4, 2] area (GE)</th>
<th>TI+ [7, 4, 3] area (GE)</th>
<th>TI+ [8, 4, 4] area (GE)</th>
<th>area latency (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKINNY Enc 128</td>
<td>64</td>
<td>5186</td>
<td>1.41</td>
<td>11974</td>
<td>15200</td>
<td>17936</td>
<td>24901</td>
</tr>
<tr>
<td>CRAFT Enc&amp;Dec</td>
<td>64</td>
<td>6000</td>
<td>1.40</td>
<td>12273</td>
<td>15570</td>
<td>18389</td>
<td>25479</td>
</tr>
<tr>
<td>CRAFT Enc Tweak</td>
<td>64</td>
<td>6121</td>
<td>1.41</td>
<td>12482</td>
<td>15827</td>
<td>18862</td>
<td>26733</td>
</tr>
<tr>
<td>CRAFT Enc&amp;Dec T.</td>
<td>64</td>
<td>6355</td>
<td>1.42</td>
<td>12788</td>
<td>16198</td>
<td>19308</td>
<td>27313</td>
</tr>
</tbody>
</table>
### 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 40nm commercial ASIC library.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SKINNY Enc</td>
<td>128</td>
<td>37</td>
<td>2041 1.28 4835 3.50 6118 3.56</td>
<td>7906 3.84 9714 4.15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LED Enc</td>
<td>128</td>
<td>49</td>
<td>1940 3.98 5286 4.87 6314 4.99</td>
<td>8197 5.46 10638 6.57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIDORI Enc</td>
<td>128</td>
<td>17</td>
<td>1616 3.31 4947 4.86 4896 4.86</td>
<td>6658 4.85 8994 5.83</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRESENT Enc</td>
<td>128</td>
<td>32</td>
<td>2050 1.04 5369 3.51 6807 3.56</td>
<td>8949 3.87 11221 4.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GIFT Enc</td>
<td>128</td>
<td>29</td>
<td>1848 0.86 5066 3.44 6331 3.52</td>
<td>8352 3.79 10701 3.92</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMON Enc</td>
<td>128</td>
<td>45</td>
<td>1984 0.93 4735 3.14 6039 3.23</td>
<td>7820 3.51 10600 4.30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PICCOLO Enc</td>
<td>128</td>
<td>32</td>
<td>1631 3.21 4648 5.52 5712 5.56</td>
<td>7662 6.14 10151 7.18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KATAN Enc</td>
<td>80</td>
<td>762</td>
<td>1236 1.60 3853 3.42 4844 3.82</td>
<td>6543 4.05 9158 4.51</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KTANTAN Enc</td>
<td>80</td>
<td>762</td>
<td>710 1.60 2696 3.80 3289 3.77</td>
<td>4242 4.08 5794 4.71</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRAFT Enc</td>
<td>128</td>
<td>32</td>
<td>1091 1.66 3041 3.34 3740 3.57</td>
<td>4961 3.78 6668 4.24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRAFT Enc&amp;Dec</td>
<td>128</td>
<td>32</td>
<td>1246 2.08 3327 3.69 4077 3.74</td>
<td>5363 3.68 7149 4.50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRAFT Enc Tweak</td>
<td>128</td>
<td>32</td>
<td>1355 1.81 3548 3.51 4368 3.69</td>
<td>5587 3.96 8508 4.45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRAFT Enc&amp;Dec T.</td>
<td>128</td>
<td>32</td>
<td>1529 2.02 3833 3.96 4705 3.62</td>
<td>6283 3.85 8983 4.73</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SKINNY Enc</td>
<td>192</td>
<td>41</td>
<td>2592 1.33 5982 3.53 7658 3.58</td>
<td>9930 3.86 12105 4.17</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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 40nm commercial ASIC library.

<table>
<thead>
<tr>
<th>Algorithm Key</th>
<th>clock cycles</th>
<th>only TI area latency</th>
<th>TI+ $[5,4,2]$ area latency</th>
<th>TI+ $[6,4,2]$ area latency</th>
<th>TI+ $[7,4,3]$ area latency</th>
<th>TI+ $[8,4,4]$ area latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRAFT Enc</td>
<td>128</td>
<td>64</td>
<td>5786 1.41 13580 3.68 17354 3.51</td>
<td>21226 3.83 28339 4.44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRAFT Enc&amp;Dec</td>
<td>128</td>
<td>64</td>
<td>6000 1.40 13888 3.96 17721 3.51</td>
<td>21097 4.19 29921 4.79</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRAFT Enc Tweak</td>
<td>128</td>
<td>64</td>
<td>6121 1.41 14088 3.82 17982 3.60</td>
<td>22150 3.83 31178 4.64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRAFT Enc&amp;Dec T.</td>
<td>128</td>
<td>64</td>
<td>6335 1.42 14393 4.11 18352 3.72</td>
<td>22620 3.89 31761 5.10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>