Concurrent Error Detection Revisited
Hardware Protection against Fault and Side-Channel Attacks

Jan Richter-Brockmann\textsuperscript{1}, Pascal Sasdrich\textsuperscript{1}, Florian Bache\textsuperscript{1} and Tim Güneysu\textsuperscript{1,2}

\textsuperscript{1} Ruhr-Universität Bochum, Horst-Görtz Institute for IT-Security, Germany
\textsuperscript{2} DFKI, Germany
firstname.lastname@rub.de

**Abstract.** Fault Injection Analysis (FIA) and Side-Channel Analysis (SCA) are considered among the most serious threats to cryptographic implementations and require dedicated countermeasures to ensure protection through the entire life-cycle of the implementations.

In this work, our contribution is twofold. First, we present a novel orthogonal layout of linear Error-Correcting Codes (ECCs) to adjust classical Concurrent Error Detection (CED) to an adversary model that assumes precisely induced single-bit faults which, with a certain non-negligible probability, will affect adjacent bits. Second, we combine our orthogonal error correction technique with a state-of-the-art SCA protection mechanism to demonstrate resistance against both threats.

Eventually, using AES as a case study, our approach can correct entirely faulted bytes while it does not exhibit detectable first-order side-channel leakage using 200 million power traces and Test Vector Leakage Assessment (TVLA) as state-of-the-art leakage assessment methodology. Furthermore, our hardware implementations reduce the area and resource consumption by $14.9\% - 18.3\%$ for recent technology nodes (compared to a conventional CED scheme).

**Keywords:** FIA · CED · ECC · SCA · LMDPL · Combined countermeasure

### 1 Introduction

Physical and implementation attacks, such as Fault Injection Analysis (FIA) and Side-Channel Analysis (SCA), have gained more and more attention by academic and industrial research over the last two decades. Rather than addressing flaws in cryptographic schemes to extract and reveal secret information, these attack vectors exploit vulnerabilities and flaws in physical implementations of algorithms considered to be cryptographically secure.

In general, FIA can be classified as set of active attacks where an adversary aims to disturb the cryptographic operation processing sensitive data and produce incorrect results that can reveal secret and sensitive internals [BS97]. Until today, a plethora of different attacks have been proposed, e.g., using clock or voltage glitches [ADN+10, ZDCT13], or focused photon injection using laser beams [SBHS15, SFG+16]. Due to the advancing in more cost-efficient equipment and more experienced adversaries, protection against FIA eventually started to gain more attention in recent years. Common approaches to increase protection against FIA mainly follow the concepts of (concurrent) error detection, error correction, or recently proposed, infective computation [GST12]. In particular, Concurrent Error Detection (CED) based on redundancy in time or area (often in terms of error detecting codes) has been researched extensively and established as the main principle to design and implement FIA countermeasures.
In contrast, SCA, can be classified as a set of passive implementation attacks that analyze the dependencies of processed sensitive information in observable physical characteristics, e.g., power consumption [KJJ99] of the device that runs the cryptographic implementation. Through statistical analysis of side-channel measurements, an adversary might be able to extract the sensitive information from the device. Due to the complexity of this approach, many attacks with different capabilities and complexities have been proposed and as a consequence, a similar wide range of countermeasure variations have been developed. In particular, masking (based on secret sharing concepts) is a promising approach due to its sound theoretical foundation and examination.

However, since techniques to thwart SCA and FIA usually follow design strategies that often are incompatible, combining both countermeasures in the same design is a challenging task. In particular, ensuring that the interplay of both approaches does not reduce the security of the device, requires careful implementations of the different mechanisms.

**Contribution.**

Our contribution in this work is twofold: In a first step, we present an alternative approach to enhance the correction capabilities of classical Concurrent Error Correction (CEC). In particular, we refrain from applying a traditional, word-oriented encoding to a cryptographic design but choose an orthogonal pattern for the linear Error-Correcting Code (ECC) instead. Benefiting from the fact that each bit of one data word will be encoded by different code words, we can correct multiple and adjacent bits of a single word which is not possible using a traditional layout. Assuming that decreasing technology sizes pose a huge challenge to adversaries precisely injecting single-bit faults and faulting adjacent bits becomes more likely, our approach will be at a clear advantage over classical schemes demonstrated in a dedicated case study.

As a second step, we extend our design to validate our claims with respect to resistance against SCA by modifying the basic architecture following on the principles of Boolean masking. This second case study then demonstrates the effectiveness of our new scheme in combination with LUT-Masked Dual-rail with Precharge Logic (LMDPL) as the SCA countermeasure of choice to resist both FIA and SCA. In particular, we use state-of-the-art leakage assessment strategies based on Test Vector Leakage Assessment (TVLA) to demonstrate that our design does not exhibit detectable leakage using up to 200 million power measurements while providing error correction capabilities at the same time.

**Related Work.**

Since the introduction of Differential Fault Analysis (DFA) in 1997 [BS97], researchers aimed to improve fault-injection attacks and countermeasures likewise [Gir04, AMT13]. Using statistical methods under the assumption of a non-uniform fault distribution, Statistical Fault Attack (SFA) [FJLT13] becomes a powerful tool to break many cryptographic implementations. Dobraunig et al. recently proposed Statistical Ineffective Fault Analysis (SIFA) [DEK+18], a combination of SFA with Ineffective Fault Attacks (IFAs) [Cla07] which has been presented as an attack that is capable to break implementations even in the presence of sophisticated countermeasures against FIA.

With advancing attack techniques, new countermeasures were proposed at same pace. In [BECN+06], various methods reaching from simple duplication to more advanced schemes applying Multi Duplication with Comparison (MDC) structures have been presented. Since simple duplication schemes are limited in security, recent publications deal with the more sophisticated application of linear ECCs [SMG16, AMR+19, SRM19] or even non-linear robust codes [RNK18] to address more advanced adversary models. In addition, to the best of our knowledge, up to now only three approaches [BKHL19, SJR+19, SRM19] are known to prevent successful SIFA by changing the fault distribution.
Research investigating the properties of combined countermeasures (against SCA and FIA) gained more interest in recent years. To address this problem, Schneider et al. [SMG16] first applied ECCs to Threshold Implementations (TIs). Later, Reparaz et al. [RDMB+18] used techniques from Multi-Party Computation (MPC) to simultaneously implement protection against SCA and FIA. Recently, De Meyer et al. [DMAN+19] combined masking with Message Authentication Code (MAC) tags originating from information theory to extend the protection against FIA.

Outline.

The remainder of this work is organized as follows: Section 2 summarizes fundamental background information on FIA. In Section 3 we outline our design considerations and describe basic design principles. Section 4 is dedicated to our first case study which applies our novel design concept to a hardware implementation of AES while Section 5 extends the basic design and architecture by SCA protection capabilities using LMDPL. Before concluding this work in Section 8, we evaluate and compare our proposals in terms of performance in Section 6 and security in Section 7.

2 Preliminaries

In the following section, we briefly introduce the considered adversary model and justify the capabilities and limitations of this adversary. Before we outline our basic design concept in the next section, we first recap the fundamental details on FIA, including the foundations of linear ECCs and the concept of CED.

2.1 Adversary Model

In recent years, it has been shown that many cryptographic implementations can be broken using various types of side channels or precise fault injections [KJJ99, ADN+10, ZDCT13, SA02, RSDT13, CLMFT14, SBHS15]. In case of side-channel attacks, we assume an adversary based on the t-probing model [ISW03] who can observe the power consumption or the electro-magnetic radiation of a device during the algorithm execution. Considering FIA, not only precisely injected single-bit faults can be used to recover secret information but instead many results also confirm that even arbitrarily injected faults can succeed to exploit such vulnerabilities [AM11, DEK+18, SJR+19]. Furthermore, common fault injection techniques, e.g., using electromagnetic pulses or clock glitches, cause sampling faults [DLM19, ADN+10]. Hence, such attack vectors do not tamper the combinatorial logic of a digital circuit but rather disturb the sampling process of a flip-flop or decrease the clock period leading to wrong values stored in the register cell. Additionally, considering continuously shrinking geometry sizes for Integrated Circuits (ICs), [SBHS15] investigates the precision of single-bit faults based on laser fault injection. The findings show that precisely injecting faults into advanced technology nodes is becoming more challenging, the smaller the geometry gets. In particular, the injection of faults in register cells becomes highly dependent on adjacent cells and their current value. To this end, we assume an adversary that is capable to precisely inject single-bit faults but with a certain, non-negligible probability will affect and change adjacent and related cells (belonging to the same part of the internal state). Moreover, since attack vectors like laser [SHS16] or electromagnetic fault injections are not limited to a single laser or coil, we also consider cases where an adversary has two synchronized lasers or coils available for precise fault injections at two different locations or points in time. For these reasons, we follow the
Well-known from communication theory, linear ECCs have been established as technique to detect and correct errors occurring on unreliable communication channels. To this end, a $k$-bit message is transformed into an $n$-bit codeword that is eventually transmitted over the (unreliable) communication channel. Hence, linear ECCs are commonly denoted as $[n,k]$-codes. Following the notations of [vT04,MS77], this paragraph briefly describes and summarizes important properties of linear ECCs relevant to this work.

A linear $[n,k]$-code $C$ of length $n$ and dimension $k$ is defined as linear subspace of $\mathbb{F}_q^n$ and is generated by a matrix $G \in \mathbb{F}_q^{k \times n}$. Hence, a message $m \in \mathbb{F}_q^k$ is encoded to its corresponding codeword $c \in \mathbb{F}_q^n$ by $m \cdot G = c \in C$. The generator matrix $G$ is the kernel of a parity-check matrix $H \in \mathbb{F}_q^{(n-k) \times n}$. As a result, $\forall c \in C$ hold $s = H \cdot c^T = 0$ where $s \in \mathbb{F}_q^{n-k}$ is called syndrome.

Note, in this work we only consider a field $\mathbb{F}_2$ with two elements since it is best suited for applications on symmetric block ciphers implemented in digital hardware circuits.

Additionally, the minimum distance $d$ of a linear code $C$, defined as the smallest Hamming distance (HD) between all codewords

$$d = \min \{|\text{HD}(c_1, c_2)|c_1, c_2 \in C, c_1 \neq c_2\} ,$$

is an essential property of any linear ECC since it determines error detection and correction capabilities. Hence, such codes are commonly called $[n,k,d]$-codes and can either detect $u = d-1$ errors or correct $v = \left\lfloor \frac{d-1}{2} \right\rfloor$ errors. If $d$ is even, $C$ can simultaneously detect $u = \frac{d}{2}$ errors and correct $v = \frac{d-2}{2}$ errors. A faulted codeword $c' = c \oplus e$, where $e$ denotes an error vector, can be corrected by an $[n,k,d]$-code as long as $\text{HW}(e) \leq v$.

Finally, a code $C$ is called a systematic code if and only if $G = [I_k | P]$ where $I_k$ denotes the identity matrix of size $k$. Note, that every generator matrix $G$ of a non-systematic linear error code $C$ can be transformed to a generator matrix $\tilde{G}$ of a systematic code with the same minimum distance $d$ [Bla03].

2.3 Principles of Concurrent Error Detection

Concurrent Error Detection (CED) is a generic concept that can be applied to any target functions to achieve fault tolerance (i.e., a correct functionality even under fault occurrence). In particular, CED allows to continuously monitor the execution of the target function and detects all covered faults during operation.

Fig. 1 outlines the main principle of CED based on spatial redundancy. A naive instantiation of this principle uses duplication of the target function for the concurrent prediction with $f_P$ instantiated as the identity function and the message passed unmodified to the prediction. Comparing the output of target function and prediction then allows
to detect any fault occurrence as a mismatch. Depending on the targeted security level, this scheme can be extended beyond duplication by instantiating multiple functions in parallel or computing more complex functions on \( f_P \), e.g., parity bits for parts or even the entire state [KKG03, BBK+03]. To this end, a more sophisticated approach applies systematic linear codes still using an unmodified target function but instead of duplication just processes the encoded part of the code for concurrent prediction. Also, if the code is chosen such that the minimum distance is \( d \geq 3 \), the capabilities of the code will be extended from detection to correction allowing to correct one (or more) faults per code word and continue operation successfully.

Eventually, the complexity of the concurrent prediction depends on the target function and the chosen code. In general, using linear ECCs, linear functions are easier to encode than non-linear functions and in case choosing \( n = i \cdot k \) with \( i \in \mathbb{N} \) and \( i \geq 2 \), it will be possible to predict the output without knowledge of \( m \). For more details, we refer the interested reader to [AMR+19, SRM19].

3 Design Concept

In this section, we first outline our basic design considerations and argue for our final code selection before we discuss the application of linear ECCs and challenges when combining our concept with SCA countermeasures.

3.1 Design Considerations

In general, our concept addresses efficiency (i.e., area and performance) and security of hardware platforms while mostly disregarding software implementations.

Given our adversary model, we implement a correction mechanism for actively injected faults instead of simple fault detection, since reliability and fault tolerance are important features of modern cryptographic implementations. However, adequate coverage of expected faults (based on the distribution in Eq. 1) is only possible when revisiting the conventional application of ECCs.

For classical encoding schemes, the correction within a single data word is limited by the Hamming distance of the ECC and the parameter \( v \). An adversary injecting faults that affect adjacent bits (e.g., due to shrinking geometry sizes) rapidly exceeds the correction capabilities of classical schemes. Since increasing the correction capabilities of the ECC easily raises the costs to an unacceptable level, our novel orthogonal design approach allows to keep the area footprint as small as possible.

Additionally, we do not only consider the protection against FIA but also address combined resistance against fault injections and (power) side channels. Selecting appropriate masking schemes for protection against SCA, in particular the transformation of non-linear sub-functions becomes more challenging and more expensive. For this reason, we decided to focus on linear and systematic codes instead of robust codes, as robust codes introduce additional non-linearity which hampers the efficient implementation of mask-based protection mechanisms against SCA.
### 3.2 Code Criteria

The detection and correction capabilities eventually depend on the choice of the parameters \( n, k, \) and \( d \), while improved capabilities directly increase the area overhead in hardware. Hence, it is essential to carefully consider the following steps for the final choice of the parameters to balance capabilities and expenses:

1. **Selection of \( k \):** For traditional schemes, \( k \) is often deduced from the word size within the *target function* but following the orthogonal encoding scheme (cf. Section 4) also allows other choices for \( k \). In fact, \( k \) is only required to be a divisor of the state size to ensure efficient, non-overlapping encodings using \( P \).

2. **Selection of \( n \):** To allow an independent data path for the redundancy \( r \) according to the CED principle (cf. Section 2.3), the codeword length \( n \) should be selected such that \( n = i \cdot k \) with \( i \in \mathbb{N} \) and \( i \geq 2 \) where \( i \) determines the desired security level and the corresponding implementation overhead.

3. **Selection of \( d \):** Given \( n, k \), the selection of the generator matrix \( G \) should be made with regard to maximize the minimal distance \( d \). Given \( n, k, d \), the construction of the final generator matrix \( G \) should be based on systematic codes with a non-singular part \( P \). This ensures the efficient inversion of internals at any point in time.

### 3.3 Correction

This section briefly describes the correction mechanism when using linear ECCs. We denote a faulty message by \( m' = m \oplus e_1 \) and a faulty redundant part by \( r' = r \oplus e_2 \), where \( e_1 \) and \( e_2 \) represent error vectors with \( \text{HW}(e_1) + \text{HW}(e_2) \leq v \) to allow correction of the occurred faults. To calculate the syndrome \( s \), the faulty message and redundancy are concatenated to \( c' = [m'|r'] \) inherently providing a concatenation of the error vectors as \( e = [e_1|e_2] \). Eventually, the syndrome is derived by

\[
s = H \cdot e^T = H \cdot (c \oplus e)^T = H \cdot e^T. \tag{3}
\]

This formula points out that all possible errors are encoded by the parity check matrix \( H \). Additionally, since we are limited by the number of correctable errors due to the code parameter \( d \), the fault-free case and all syndromes with the corresponding faults can be perfectly stored in a Look-Up Table (LUT) with \( S \) entries where

\[
S = 1 + \sum_{i=1}^{v} \binom{n}{i} \tag{4}
\]

Generally, for a valid code word \( c \) holds \( 0 = H \cdot c^T = H \cdot G^T \cdot m^T \) which implies \( H \cdot G^T \perp 0 \). Since we specified to use systematic codes, \( H \) can be simply set to \( H = [P^T|I_{n-k}] \). It turns out that the calculation of the syndrome simplifies to an encoding of \( m' \) by \( P^T \) and a subsequent addition (modulo two) with \( r' \).

Additionally, the position of the correction modules in the target cipher should be well-considered. First, a correction module should be placed before every *non-linear function*. If an erroneous word is fed into a non-linearity, the error propagation will be hardly predictable and could eventually prevent the correction. Second, the inputs of any function that combines code words to produce a new code word should be fault free. Otherwise faults from different sources can be accumulated and a correction might become unfeasible.
3.4 Combination with SCA Countermeasures

In this section we identify challenges and requirements that should be considered when combining our technique with protection mechanisms against SCA.

**Sequential logic:** We assume faults occurring with a higher probability in sequential logic (since effective faults eventually manifest in registers). Hence, our approach focuses on protection of the data-dependent flip-flops whose number should be minimized for better efficiency of the correction modules.

**Non-linear functions:** Due to our rearranged scheme, all code words are generated from \( k \) bits of different words and each non-linear function \( F: \mathbb{F}^w \rightarrow \mathbb{F}^w \) has to be combined with a preceding decoding \( f_P^{-1} \) and a succeeding encoding \( f_P \) which introduces overhead with every additional non-linear function between register stages.

**Hardware primitives:** Since our approach protects all data dependent registers, we cannot apply countermeasures based on hardware primitives or macros without direct access to register values (e.g., given for BRAM modules in modern Field-Programmable Gate Arrays (FPGAs)).

4 Case Study A: FIA Countermeasure

This section presents a practical case study using the Advanced Encryption Standard (AES) and discusses important features of our hardware design.

4.1 Encoding Procedure

As mentioned in Section 3.2, our implementation is not restricted to the underlying functions of AES, hence we choose \( k = 4 \) and encode the AES state matrix in an orthogonal orientation to the state bytes. The encoding is performed on \( [A_i, A_{i+4}, A_{i+8}, A_{i+12}] \) where \( A_i \) denotes a single byte and \( A_i^j \) a single bit of the corresponding byte with \( 0 \leq i \leq 3 \) and \( 0 \leq j \leq 7 \). As an example, the encoding of the first row of the AES state matrix is shown in Fig. 2. For each row the encoding scheme produces eight redundant nibbles \( r_j \) which are processed by a concurrent predictor. To this end, the approach requires \( 4 \cdot 8 = 32 \) generators \( P \) in total to encode the entire 128-bit state in parallel.

In contrast to our chosen encoding scheme applied on single bits out of the same row, it is also conceivable to apply it to single bits out of the same column. Generally this approach can be realized with the same error coverage as for the introduced scheme based
Concurrent Error Detection Revisited

on a row-encoding. However, since \texttt{ShiftRows} works on the AES state matrix’s rows, a column aligned design would increase complexity and hence result in an increased area consumption.

4.2 Code Selection

Given $k = 4$ and the fact that we opted for $n = 2k$, we performed an exhaustive search over all potential matrices $P$ that provide a minimum Hamming distance $d = 4$ and are invertible\(^1\). It turns out that the matrix

$$P = \begin{bmatrix}
1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1
\end{bmatrix}$$

and all $4! = 24$ possible arrangements of these four rows and columns are the only matrices that fulfill our requirements. However, because the above stated matrix $P$ is also self-inverse, we decide to use it for our countermeasure as it simplifies the implementation process.

A direct consequence of choosing a self-inverse matrix $P$ is that instead of using the parity check matrix $H = [P^T \vert I_k]$, we can apply the inverse of the transposed $P$ to the redundant part such that $H$ can be adapted to \(\tilde{H} = [I_k \ (P^T)^{-1}] = [I_k | P] = G\) while the syndrome $s$ is still zero $\forall c \in C$. As explained below, this method allows particular optimizations when implementing the predicting module for \texttt{SubBytes}. Note, however, that a different LUT is required to correct the corresponding faults although the fault coverage is left unaltered.

4.3 Concurrent Prediction

For concurrent prediction of the AES encryption using the orthogonal encoding scheme, all sub-functions have to be reconsidered before application to the redundant part of the internal state. In particular for the byte-oriented, non-linear sub-functions, application of the orthogonal encoding within the concurrent prediction becomes more challenging and requires careful consideration.

\textbf{Key Addition.} Predicting the output of the key addition is straightforward and requires no modification of the addition module, assuming that the round key encoding follows the same orthogonal encoding principle as the state. Due to the linearity of the chosen code and the key addition operation, the addition prediction can be performed concurrently on the redundancy using the same addition operation (XOR).

\textbf{Shifting of Rows.} Fortunately, the predictor for \texttt{ShiftRows} also comes without any additional costs mainly due the inherent structure of the chosen code. All left-shifting of each particular row can be applied independently of the remaining rows and performed bit-wise such that each word \([A_i^j \ A_i^{j+4} \ A_i^{j+8} \ A_i^{j+12}]\) can be considered independently. Hence, the successive operations of decoding, left-shifting, and encoding given as $P \circ \text{LS} \circ P^{-1}$ can be combined into a single operation. In particular, given the previously chosen $P$

\(^1\)We decided to choose a code with a minimum distance $d = 4$, since, in addition to the 1-bit error correction, this also provides capabilities for 2-bit error detection.
eventually leads to
\[
\begin{pmatrix}
0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0
\end{pmatrix}
= \begin{pmatrix}
1 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0
\end{pmatrix}
\cdot \begin{pmatrix}
0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1
\end{pmatrix}
\]
which can be implemented as simple right-shifting of each particular row (instead of the original left-shifting).

**Byte Substitution.** Applying a conventional encoding scheme to `SubBytes` could be realized by merging the preceding decoding step with the actual execution of the substitution and a subsequent encoding into an adapted S-box. Assuming an implementation of `SubBytes` as LUT, would require the same hardware resources for both realizations. However, in case of our applied orthogonal encoding scheme, we are not able to merge these three steps into one recomputed LUT as every S-box input depends on eight different codewords. Instead, we can optimize our implementation by adapting the preceding correction module using \(\hat{H}\) (cf. Section 4.2) such that the inputs to our modified substitution layer are already decoded during the correction process. Hence, no dedicated initial decoding is required and the adapted module just contains the original S-box and a successive orthogonal encoding.

**Mixing of Columns.** For a conventional encoding scheme \(g_P\) the predictor of `MixColumn` would be realized by a preceding decoding of the redundancy \(r\) and re-encoding after executing the actual operation as depicted in Eq. 5.

\[
MC' = g_P(MC(g_P^{-1}(r)))
\]

Obviously, these three operations do not have to be performed separately but rather can be merged into one modified `MixColumn` \(MC'\). This would reduce additional hardware costs but still exceed the resources compared to an unmodified implementation.

However, our proposed encoding scheme does not require any modifications of `MixColumn` to predict the output and can be used unaltered. This benefit comes from the fact that \(f_P(MC(A)) = MC(f_P(A))\) holds for the applied orthogonal encoding function \(f_P\) and any arbitrary \(P\). Each bit of a single byte is multiplied by the same bit of the encoding matrix \(P\) and it does not matter if the multiplication happens before or after a multiplication in \(GF(2^8)\).

### 4.4 Implementation Overview

Fig. 3 shows a schematic overview of our proposed implementation. The left part represents the encryption path and is left unaltered compared to a parallel, round-based implementation of AES (except for the inserted correction module). We decided to use a two-stage pipeline version to correct faults before `SubBytes` and `MixColumn` to prevent undesired fault propagation (cf. Section 3.3).

The right part represents the prediction circuit where the plaintext is initially transformed by \(P\) to provide the redundant parts of the code words. As mentioned before, the execution of the key addition requires another encoding module to encode the corresponding round key prior to the addition. The predicting sub-function of `SubBytes` \(SB'\) contains the original AES S-box followed by an encoding matrix \(P\) as explained in Section 4.3. As a first step of the diffusion layer, the \(SR'\) module predicts the `ShiftRows` operation performing right-shifting instead of left-shifting as explained in Section 4.3. Before the second step of the diffusion layer prediction is performed using an unaltered `MixColumns`
module, the intermediate result of the ShiftRows prediction is buffered in a register stage and used for the correction inside the shared correction module.

Also, we would like to emphasize that Fig. 3 only shows an implementation with a single and shared correction module in the data path. In general, this strategy provides a realization with smaller area overhead but reduced throughput. A doubling of the throughput can be achieved when spending more area and implementing two independent correction modules, one per register stages (located at the outputs of the registers) as the design allows pipelining.

4.5 Correction Module

Above, we mentioned that the decoding step before applying SubBytes can be merged with the operations in the correction module by modifying the correction process from Section 3.3 using $\tilde{H}$ as parity check matrix. In case of implementing our approach using only one correction module, it would be desirable to apply the same LUT for correcting faults after AddKey and ShiftRows in order to keep the footprint low. To this end, the decoding is done by multiplying the redundancy $r_{\text{in}}$ with $P$ as explained above and depicted in Fig. 4. To determine the corresponding error vector $e = [e_1 | e_2]$, the input data $m_{\text{in}}$ and decoded redundancy $\tilde{r}_{\text{in}} = P \cdot r_{\text{in}}$ are added and afterwards fed into the LUT. The correction in the encryption path can be accomplished by adding the first part of the error vector $e_1$ to $m_{\text{in}}$. For the redundancy we consider three cases:

1. The correction is done after AddKey such that the outputs $m_{\text{out}}$ and $r_{\text{out}}$ serve as inputs to SubBytes and a fault has not occurred in the redundant path ($\text{FAULTY}_R = 0$ and $\text{CORRECT}_SB = 1$)

2. The correction is done after AddKey such that the outputs $m_{\text{out}}$ and $r_{\text{out}}$ serve as inputs to SubBytes and a fault occurred in the redundant path ($\text{FAULTY}_R = 1$ and $\text{CORRECT}_SB = 1$)

3. The correction is done after ShiftRows such that the outputs $m_{\text{out}}$ and $r_{\text{out}}$ serve as inputs to MixColumns ($\text{FAULTY}_R = X$ and $\text{CORRECT}_SB = 0$)

While $\text{CORRECT}_SB$ can be easily determined by tracking the current position of valid data in the algorithm flow, $\text{FAULTY}_R$ is only true if the error vector $e_2$ is non-zero. In
Case, both control signals are in a true state, a fault occurred in the redundant path and the corrected output should be returned for processing in SubBytes. Hence, the already decoded redundant input needs to be corrected and fed to the output to get rid of the decoding module before applying the substitution. As the LUT is generated for correcting faults on the input data \( m_{in} \) and \( r_{in} \), the actual error vector \( e_2 \) has to be transformed so that it can be used to correct faults on the decoded redundant data. Due to the advantageously chosen matrix \( P \), this transformation can be realized by reversing and subsequently negating \( e_2 \) as for all one-bit error-vectors \( e_{one} \) the matrix \( P' \) describes a flip of every bit (cf. Eq. 6).

\[
\bar{e}_{one} = P^{-1} \cdot e_{one} = \begin{pmatrix}
1 & 1 & 0 & 0 \\
1 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1
\end{pmatrix}
\cdot
\begin{pmatrix}
0 & 1 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 \\
1 & 1 & 1 & 0
\end{pmatrix}
\cdot
\begin{pmatrix}
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0
\end{pmatrix}
\cdot
\begin{pmatrix}
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 1 & 0
\end{pmatrix} \cdot e_{one}
\]

(6)

Finally, the reversed and negated error vector \( \bar{e}_2 \) can be added to the decoded redundancy \( \tilde{r}_{in} \) and afterwards can be used in SubBytes in the redundant path.

In case no fault occurred in the redundant path, \( e_2 \) is the zero-vector and the decoded redundancy is released to the output \( r_{out} \). The case where CORRECT_SB is zero can be easily covered without using the output of the multiplexer controlled by FAULTY_R as shown in Fig. 4. The error vector \( e_2 \) is directly routed to an XOR-operation adding a zero-vector in case no fault occurred or removing an error in \( r_{in} \) in case \( e_2 \) is non-zero.

5 Case Study B: Combined Protection

We now extend our approach by adding a countermeasure against SCA. Before we describe the combined design, we briefly justify the selection of the chosen masking scheme.

5.1 Countermeasure Selection

In Section 3.4 we defined three requirements for an efficient combination of our proposed orthogonal encoding scheme with a countermeasure against SCA. Based on this list, we decided to implement the LMDPL scheme from [LMW14, SBHM20]. LMDPL is a masking technique which was specially designed to avoid glitches and early propagation. The implementation is realized on gate-level such that each elementary gate of a target function can be replaced by a protected gate. Therefore, the structure of a cipher can be left unaltered and non-linear functions do not have to be separated by several pipelining stages. This perfectly matches our first and second requirement avoiding additional register stages and additional non-linearities. Furthermore, LMDPL requires registers around a non-linear
function $\mathcal{F}$ to allow precharging its combinatorial logic. Since our design already includes a pipeline stage before and after \texttt{SubBytes}, LMDPL seems to be a natural fit. Finally, LMDPL is a generic approach and does not require any platform specific primitives (e.g., BRAM in FPGAs) which meets our last requirement. All register outputs are directly accessible and can be routed through a correction module introduced in Section 4.5. Even though LMDPL does not allow to use pipelining, we decided to avoid the routing to the correction module through the multiplexers (cf. Fig. 3) and instantiated two correction modules right after both register stages.

5.2 Combined Approach

Since LMDPL is based on masking, both the encryption and the redundant path have to be modified and split into masked shares to protect our proposed FIA countermeasure against SCA. Fig. 5 shows the realization of \texttt{SubBytes} as it is the most interesting part and the only part equipped with dual-rail logic. All in all, we get a design with four different data paths. The leftmost one processes the intermediate results of the data share while the path right beside generates new table values from $m$. To generate updated table values for all 16 S-boxes in parallel, the design requires 576 bit fresh randomness every two clock cycles provided by a Pseudorandom Number Generator (PRNG) based on KECCAK [BDPVA13]. Like the mask $m$ this randomness is reused in the redundant part containing our proposed scheme. Both the data and the mask share are protected against FIA by the rightmost module and the second rightmost, respectively. To avoid any demasking in the correction process, each share is corrected separately on the masked data.

One important key feature of LMDPL is the precharge phase of combinatorial logic placed in the S-boxes. In the original work this precharge phase is realized by resetting the input and table registers such that all gates and signals will be forced to zero. However, our orthogonal encoding scheme includes corrections modules right after every data dependent register. Even if our design would reset these registers, this would still result in glitches due to the optimized correction module. Hence, we realized the precharge phase by a multiplexer which either precharges the LMDPL logic with zeros or forwards the current state. To avoid any glitches and ensure that all input signals to the multiplexer are settled, we triggered the precharge signal by the falling edge of the clock. Furthermore, this modification allows us to remove the high amount of registers storing the table values. The mask share is now synchronized with the data share and the computed table values can be used in the same clock cycle at the falling edge of the clock. For a more optimized implementation regarding the critical path a second clock domain (e.g., a faster or phase shifted clock) could be used instead.
Table 1: FPGA implementation results (xcku035).

<table>
<thead>
<tr>
<th>Resources</th>
<th>Logic</th>
<th>Memory</th>
<th>Area</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LUT</td>
<td>FF</td>
<td>CLB</td>
<td>Freq.</td>
</tr>
<tr>
<td>Reference AES Implementation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Round-Based</td>
<td>1388</td>
<td>271</td>
<td>253</td>
<td>454.5</td>
</tr>
<tr>
<td>Classical CED with Correction</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>One Corr. Module</td>
<td>4533</td>
<td>659</td>
<td>714</td>
<td>259.7</td>
</tr>
<tr>
<td>Two Corr. Modules</td>
<td>5756</td>
<td>659</td>
<td>911</td>
<td>303</td>
</tr>
<tr>
<td>Orthogonal CED with Correction (OCEC)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>One Corr. Module</td>
<td>3532</td>
<td>659</td>
<td>583</td>
<td>277.8</td>
</tr>
<tr>
<td>Two Corr. Modules</td>
<td>3717</td>
<td>659</td>
<td>589</td>
<td>339</td>
</tr>
<tr>
<td>Combined Approach</td>
<td>31141</td>
<td>1177</td>
<td>4290</td>
<td>46.1</td>
</tr>
</tbody>
</table>

6 Evaluation

In this section we present and discuss the implementation results for FPGAs and Application-Specific Integrated Circuits (ASICs). Moreover, we compare our approach to related work.

6.1 FPGA Implementation

For comparison reasons, we first implemented a countermeasure using linear ECCs arranged in a conventional way as we are not aware of any works that applied linear codes to AES (using a 20 nm Xilinx Kintex UltraScale FPGA). We opted for a $[16,8,5]$-code\(^2\) to align the word size of $k=8$ in AES with the generator input width. For both, the conventional and orthogonal CED, we implemented two variants optimized for either area (single correction module) or throughput (two correction modules). Note that we considered the Independence Property from [AMR+19,SRM19] for all our implementations.

Using one correction module, our approach outperforms the conventional CED scheme by 18.3% in terms of CLB resources (as shown in Table 1) since the orthogonal arrangement allows to reuse MixColumn and the area for the LUTs to correct occurring faults is considerably smaller. Similarly, when adding a second correction module, the required CLBs for our novel design is decreased by 35.3% compared to the conventional approach.

6.2 ASIC Implementation

Table 2 provides ASIC results for three different libraries. We started our synthesis with the Open Cell Nangate 15 nm library\(^3\), but we also provide implementation results for Nangate 45 nm and UMC 90 nm libraries.

Again, our approach outperforms the conventional CED scheme by 14.9% using the NGate15 library and one correction module. For two correction modules, we highlight two interesting points. First, in the 45 nm technology the area decreases slightly compared to the design using one correction module (due to better optimization during synthesis) while

\(^2\)Such a code can be found in the appendix of [BCC+14].
\(^3\)www.silvaco.com/products/nangate/FreePDK15_Open_Cell_Library/index.html
Table 2: ASIC implementation results.

<table>
<thead>
<tr>
<th></th>
<th>NGate15</th>
<th></th>
<th>NGate45</th>
<th></th>
<th>UMC90</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Area</td>
<td>Freq.</td>
<td>T'put</td>
<td>Area</td>
<td>Freq.</td>
<td>T'put</td>
</tr>
<tr>
<td></td>
<td>kGE</td>
<td>MHz</td>
<td>MB/s</td>
<td>kGE</td>
<td>MHz</td>
<td>MB/s</td>
</tr>
<tr>
<td>Reference AES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Round-Based</td>
<td>15.5</td>
<td>4016</td>
<td>4943</td>
<td>14.2</td>
<td>870</td>
<td>1070</td>
</tr>
<tr>
<td>Classical CED with</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Correction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>One Corr. Module</td>
<td>39.9</td>
<td>2360</td>
<td>1642</td>
<td>36.2</td>
<td>490</td>
<td>341</td>
</tr>
<tr>
<td>Two Corr. Modules</td>
<td>46.7</td>
<td>3416</td>
<td>4555</td>
<td>42.4</td>
<td>625</td>
<td>833</td>
</tr>
<tr>
<td>Orthogonal CED with</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Correction (OCEC)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>One Corr. Module</td>
<td>33.9</td>
<td>1634</td>
<td>1137</td>
<td>31.4</td>
<td>543</td>
<td>378</td>
</tr>
<tr>
<td>Two Corr. Modules</td>
<td>34.2</td>
<td>3417</td>
<td>4556</td>
<td>31.2</td>
<td>787</td>
<td>1050</td>
</tr>
<tr>
<td>Combined Approach</td>
<td>112.8</td>
<td>180</td>
<td>125</td>
<td>101.7</td>
<td>128</td>
<td>89</td>
</tr>
</tbody>
</table>

providing higher throughput due to pipelining. Second, the gap between the footprints of a conventional encoding and our proposed orthogonal scheme significantly increases.

6.3 Comparison to Previous Work

In Table 3 we compare our approach to already existing countermeasures. Starting in 2002, Karri et al. proposed a countermeasure against FIA for AES which is based on the inverse operations of the encryption algorithm [KWMK02]. In 2004, Bertoni et al. suggested to use parity schemes to detect faults [BBKM04]. Although these countermeasures come with small additional implementation cost regarding the area overhead, they do not provide an appropriate security level.

Compared to [SMG16] and [DMAN+19] our design comes with roughly the same area overhead on a modern Xilinx UltraScale FPGA. In the 45 nm technology, our approach requires slightly less hardware resources. However, both approaches rely on fault-detection and withhold the faulty ciphertext or perform an infective computation, respectively. Hence, all aforementioned types of countermeasures can be broken by SIFA as shown in [DEK+18].

Only [BKHL19] and [SJR+19] provide security against SIFA. The former approach relies on modular redundancy and comes with a huge area overhead which was, however, not explicitly evaluated. The later work considers a combination of a transform-and-encode strategy while the encoding is realized by linear ECCs. Unfortunately, no target platform and no overhead were provided.

7 Security Evaluation

In this section we first discuss the fault coverage of our proposed scheme and compare it with the coverage of conventional encoding (i.e., applying a [16, 8, 5]-code aligned to bytes) and Triple Modular Redundancy (TMR). Afterwards, we evaluate the resistance of our combined countermeasures against SCA.

7.1 Fault Coverage

To evaluate the resilience against FIA, we consider the adversary model defined in Section 2.1. Generally, the fault coverage of an arbitrary protection scheme is determined
Table 3: Comparison to other countermeasures against FIA.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Cipher</th>
<th>Target Platform</th>
<th>Area Overhead</th>
<th>Correction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karri et al. [KWMK02]</td>
<td>AES(^a)</td>
<td>Xilinx XCV1000BG</td>
<td>20.97 %</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>AES(^b)</td>
<td>Xilinx XCV1000BG</td>
<td>18.90 %</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>AES(^c)</td>
<td>Xilinx XCV1000BG</td>
<td>38.08 %</td>
<td>NO</td>
</tr>
<tr>
<td>Bertoni et al. [BBKM04]</td>
<td>AES</td>
<td>STMicro 0.18 (\mu)m</td>
<td>18.00 %</td>
<td>NO</td>
</tr>
<tr>
<td>Schneider et al. [SMG16]</td>
<td>LED</td>
<td>UMC 0.18 (\mu)m</td>
<td>156 %</td>
<td>NO</td>
</tr>
<tr>
<td>De Meyer et al. [DMAN+19]</td>
<td>AES</td>
<td>NGate 45 nm</td>
<td>153 %</td>
<td>NO</td>
</tr>
<tr>
<td>Breier et al. [BKHL19]</td>
<td>GIFT-64(^i)</td>
<td>–</td>
<td>&gt; 200 %</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>GIFT-64(^ii)</td>
<td>–</td>
<td>&gt; 400 %</td>
<td>YES</td>
</tr>
<tr>
<td>Saha et al. [SJRW+19]</td>
<td>Present</td>
<td>–</td>
<td>N/A</td>
<td>YES</td>
</tr>
<tr>
<td>This Work</td>
<td>AES</td>
<td>UltraScale xcku035</td>
<td>154 %</td>
<td>YES</td>
</tr>
<tr>
<td>This Work</td>
<td>AES</td>
<td>NGate 45 nm</td>
<td>119 %</td>
<td>YES</td>
</tr>
</tbody>
</table>

\(^a\) Algorithm level. \(^b\) Round level. \(^c\) Operation level. \(^i\) Single-bit faults. \(^ii\) Double-bit faults.

by Eq. 7 where \(F_{\text{not}}\) denotes the number of faults that cannot be corrected by a target countermeasure and \(F_{\text{tot}}\) the total number of possible faults.

\[
C(b) = 1 - \frac{F_{\text{not}}}{F_{\text{tot}}}
\]

Furthermore, we do not distinguish between faults occurring in the encryption part or in the redundant part.

7.1.1 Fault Coverage assuming Faults in one Byte

Initially, we discuss the first scenario where an attacker injects faults into one single byte following a biased distribution \(E_{B_b}\) (see Fig. 6a). Our approach and a TMR based scheme can correct every possible fault. Our approach uses a \([8, 4, 4]\)-code and is able to recover one entire faulty byte due to the orthogonal layout. TMR corrects any faults that occur in just one of the copies such that a coverage of 100 % is achieved (i.e., \(F_{\text{not}}=0\)). Implementing a conventional encoding scheme for AES, leads to a decreased coverage for \(b \geq 3\) since the underlying \([16, 8, 5]\) code is only capable of correcting up to two faults. Following Eq. 7, the fault coverage \(C_{\text{conv}}(b)\) is defined by

\[
C_{\text{conv}}(b) = 1 - \frac{\sum_{i=3}^{b} \binom{8}{i}}{\sum_{i=1}^{b} \binom{8}{i}}
\]

7.1.2 Fault Coverage assuming Faults in two Bytes

Considering the second scenario of our adversary model, where an attacker is assumed to inject faults into up to two different bytes following \(E_{B_b}\), we again make use of Eq. 7. To determine the total number of possible faults \(F_{\text{tot}}(b, R)\), we used Eq. 8 where \(R\) denotes the number of bytes that can be targeted by an attacker.

\[
F_{\text{tot}}(b, R) = \sum_{i=1}^{b} \left[ \binom{2 \cdot 8}{b} - 2 \binom{8}{b} \right] \cdot \binom{R}{2} + R \binom{8}{b} + \sum_{i=1}^{2} \binom{8 \cdot R}{i}
\]

Without loss of generality, we considered only one row of the AES state matrix and its corresponding redundancy. To this end, the number of attackable bytes \(R\) results in \(R=8\) for our approach and the conventional encoding scheme and in \(R=12\) for TMR as the
scheme requires two copies of the original design. If \( b \leq 2 \), the number of faults can be determined by the second summand as only two bytes can be targeted and the faults are uniformly distributed over the attackable area. In case \( b > 2 \) the first summand has to be taken into account as well. By using the first binomial coefficient \( \binom{2^8}{b} \), we determine the number of different faults with \( b \) flipped bits within two bytes. As we would like to calculate all combinations of such faults in an entire row, we multiply it by \( \binom{8}{2} \). However, before multiplying, we subtract \( 2 \binom{8}{b} \) possible faults to exclude all cases where all bit flips \( b \) occur in one single byte which otherwise would be counted several times. These cases are added just once by \( R(\binom{8}{b}) \).

The number of uncorrectable faults \( F_{\text{not}} \) for all three cases were determined by a simulation. We iterated over all possible error vectors combined by two bytes \( x \) and \( y \) and bounded by \( b \) which results in \( \binom{16}{b} \) possibilities. Since all approaches are based on different correction capabilities, we distinguished between these cases.

1. **Our Approach** For each possibility where \( \text{HW}(x \odot y) > 0 \) we found a fault that cannot be corrected since two bit flips occurred in the same codeword (\( \odot \) denotes a bitwise and). This method gives us all possible faults that cannot be corrected considering just two bytes. To cover all cases for the entire row, we multiply the result by \( \binom{8}{2} \).

2. **Conventional Encoding** For each possibility where \( \text{HW}(x) > 2 \lor \text{HW}(y) > 2 \) we found a fault that cannot be corrected since more than two bit flips occurred in one codewords To cover all cases for the entire row, we multiply the result by \( \binom{8}{2} \) as above.

3. **TMR** For each possibility where \( \text{HW}(x) \geq 1 \land \text{HW}(y) \geq 1 \) we found \( 4 \cdot 8 + 4 \cdot 4 = 48 \) faults that cannot be corrected since they occurred in more than one instantiation. No final multiplication is required.

The resulting fault coverages are given in Fig. 6b. In case \( b=1 \), all three approaches achieve 100% fault coverage. For \( b=2 \), the coverage for TMR rapidly drops below 35% as faults only can be corrected when they occur in the same instantiation. While the conventional encoding scheme still provides 100% security (since \( d=5 \)), the correction capability of our design decreases to 89.23%. Assuming up to three faults, our approach and the conventional encoding provide similar results whereas our approach performs slightly worse. From there on, our approach outperforms the conventional encoding. TMR performs worse than the orthogonal for \( b \leq 6 \) method but has a better fault coverage for \( b \geq 7 \).
Figure 7: Confidence intervals for $\alpha = 0.01$ and 200 million traces. Lower bounds in red, upper bounds in blue.

7.2 SCA Evaluation

We applied the non-specific fixed-vs-random approach presented in [SM15] to evaluate the side-channel security of our proposal. To identify quantifiable lower and upper bounds for the leakage, we further employed the confidence-interval framework from [BPG18] where all confidence intervals are calculated for a family-wise error rate corrected significance level $\alpha = 0.01$. The plain $t$-test results and a sample trace are provided in Appendix B.

Our implementation was instantiated on a Sakura-X side-channel evaluation board where the target was supplied with a 4 MHz clock and the current was measured indirectly via the voltage drop over a shunt in the supply path (with a 20 dB low-noise amplifier) using a 8 bit oscilloscope at a sample rate of 1.25 GS/s.

The first-order evaluation of the protected design is depicted in Fig. 7a and the lower bound of zero for every sample indicates that our evaluation was not able to detect first-order leakage. If there is some (undetectable) leakage, the absolute difference in means for the measurements is lower than $0.003 \text{ LSB}$. The second-order evaluation is shown in Fig. 7b and confirms the detection of expected (albeit small) second-order leakage. For at least one sample point the absolute difference in variances is $0.027 \text{ LSB}^2$ while there is no difference above $0.033 \text{ LSB}^2$ for any sample. The third-order evaluation in Fig. 7c shows results similar to the second-order case, but the influence of noise increases exponentially in the attack and evaluation order [CJRR99], hence the confidence intervals are not as tight as in the previous case.

8 Conclusion

Revisiting Concurrent Error Detection, we presented a novel approach to use ECC for protection against active FIA. Arranging ECCs in an orthogonal pattern, we are able to correct adjacent bits of internal values using simple linear codes. Using a case study based on AES, we show that our approach is up to $35.3\%$ smaller in terms of area compared to classical arrangements while providing better fault coverage for the considered adversary.
model. Our second case study shows that our approach combines efficiently with state-of-the-art masking countermeasures to extend the protection even against passive side-channel analysis. Through practical evaluation using 200 million power traces, we validated the security of our design against first-order SCA using a state-of-the-art leakage assessment methodology.

Acknowledgments

The work described in this paper has been supported in part by the German Research Foundation (DFG) under Germany’s Excellence Strategy - EXC 2092 CASA - 390781972 and by the German Federal Ministry of Education and Research BMBF through the projects SecRec (16KIS0606K) and VeriSec (16KIS0603).

References


A Unprotected Design

Fig. 8 shows the measurement results for the first statistical moments of our combined countermeasure when disabling the random number generator providing randomness for the LMDPL masking scheme. As expected, we can clearly identify first-order leakage.

![Figure 8: 1st-Order results of the unprotected design (10000 traces).](image)

B t-Test Results

For reference, Fig. 9 shows a sample trace of the power consumption of our implementation with enabled SCA countermeasures. Fig. 10 provides the evaluation results of the same collected data as analyzed in Section 6 using Welsh’s t-test. The threshold of $t_{th} = 4.93$ is adjusted for family-wise error rate at a total significance of $\alpha = 0.01$. An unprotected design shows clear leakage using only 10000 traces as shown in Fig. 10a. Our protected implementation exhibits no first-order leakage even when 200 million traces are used in the evaluation. Fig. 10c and Fig. 10d show the second- and third-order t-test results respectively. As expected, some leakage can be detected.

![Figure 9: Example trace of the protected design.](image)
(a) 1st-Order $t$-test of the unprotected design (10,000 traces).

(b) 1st-Order $t$-test of the SCA protected design (200 m traces).

(c) 2nd-Order $t$-test of the SCA protected design (200 m traces).

(d) 3rd-Order $t$-test of the SCA protected design (200 m traces).

Figure 10: Non-specific $t$-test results for $\alpha = 0.01$. $t$-values are solid blue, the significance threshold is red.