Balanced Encoding of Near-Zero Correlation for an AES Implementation

Seungkwang Lee and Jeong-Nyeo Kim

Abstract—Power analysis poses a significant threat to the security of cryptographic algorithms, as it can be leveraged to recover secret keys. While various software-based countermeasures exist to mitigate this non-invasive attack, they often involve a trade-off between time and space constraints. Techniques such as masking and shuffling, while effective, can noticeably impact execution speed and rely heavily on run-time random number generators. On the contrary, internally encoded implementations of block ciphers offer an alternative approach that does not rely on run-time random sources, but it comes with the drawback of requiring substantial memory space to accommodate lookup tables. Internal encoding, commonly employed in white-box cryptography, suffers from a significant security limitation as it does not effectively protect the secret key against statistical analysis. To overcome this weakness, this paper introduces a secure internal encoding method for an AES implementation. By addressing the root cause of vulnerabilities found in previous encoding methods, we propose a balanced encoding technique that aims to minimize the problematic correlation with key-dependent intermediate values. We analyze the potential weaknesses associated with the balanced encoding and present a method that utilizes complementary sets of lookup tables. In this approach, the size of the lookup tables is approximately 512KB, and the number of table lookups is 1,024. This is comparable to the table size of non-protected white-box AES-128 implementations, while requiring only half the number of lookups. By adopting this method, our aim is to introduce a non-masking technique that mitigates the vulnerability to statistical analysis present in existing internally-encoded AES implementations.

Index Terms—Block cipher, AES, power analysis, internal encoding, countermeasure.

I. INTRODUCTION

In THE context of the gray-box model, when software cryptographic implementations run in untrusted environments, they become vulnerable to attacks exploiting side-channel information like timing or power consumption. This type of information often exposes secret keys, allowing attackers to retrieve them without the need for laborious reverse engineering efforts. Among the various sources of side-channel information leakage, one widely exploited aspect is the correlation with key-dependent intermediate values.

For instance, power analysis techniques, including Differential Power Analysis (DPA) [1] and Correlation Power Analysis (CPA) [2], analyze the correlation between the power consumption during cryptographic computations and hypothetical values associated with those computations. In such attacks, the attacker interacts with the software implementation of a cryptographic primitive by providing arbitrary inputs and recording power traces using an oscilloscope, a DC current or power analyzer, or other specialized electronic instrumentation. For each pair of input and subkey candidate relevant to the target cryptographic function (e.g., SubBytes), the attacker computes the hypothetical value. The correct candidate yields hypothetical values that strongly correlate with specific points in the power traces because the power consumption is directly related to the data processed in the circuit.

In DPA, power traces are grouped into distinct sets based on these hypothetical values, and average traces are generated for each set. If the subkey candidate is correct, a distinctive peak will appear in the differential trace at the points associated with the target function. In contrast, in CPA, the subkey is deduced by calculating the correlation between hypotheses and individual points in the power traces.

The number of power traces necessary for a successful power analysis primarily depends on the power model and electronic noise. A power model, such as a bit model or Hamming Weight (HW) model, defines how the power consumption of the target function is represented. On the other hand, electronic noise results from power measurement and constant components like leakage current and transistor switching [3]. Generally, as noise levels increase, the required number of power traces for key recovery also increases.

In recent years, several studies [4], [5], [6] have introduced a novel approach to computational analysis. This approach involves providing a security verifier or attacker access to either the source code or the executable of the target primitive. In this model, computational traces can be collected by observing the computations in memory without the interference of electronic noise. Consequently, key-dependent intermediate values and memory addresses related to read and write access are readily exposed.

1556-6021 © 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
The concept of computational traces was initially introduced by Differential Computation Analysis (DCA) [5]. DCA is a non-invasive attack that can extract secret keys from a wide range of different white-box implementations [7, 8] without necessitating a detailed reverse engineering of the target implementation. From the attacker’s perspective, noise-free computational traces have also facilitated DCA-variant attacks, such as zero difference enumeration [9], collision attacks [10], and bucketing attacks [11]. From the standpoint of cryptographic engineers, noise-free traces can be employed to assess the protection of secret keys against power analysis with a reduced number of traces compared to classical power analysis and without the need for electronic instrumentation that can be very expensive when attacking systems operating at high clock frequencies.

To mitigate problematic correlations in side-channel attacks, common software countermeasures include the use of masking and operation shuffling techniques. Masking relies on the concept of splitting a sensitive variable into shares, denoted as $s_0, s_1, \ldots, s_d$. These shares are combined through a group operation $\circ$ (typically XOR or modular addition) to recover the original value $s$. It is crucial that any subset of fewer than $d+1$ shares or leakage signals remains statistically independent of $s$. To achieve this, the masks $s_1, \ldots, s_d$ are randomly selected, and the masked variable $s_0$ is computed as $s_0 = s \circ s_1 \circ \ldots \circ s_d$. The parameter $d$ is referred to as the masking order. Here, it’s worth noting that even with the same order (e.g., $d = 1$), masking techniques can exhibit varying performance and security depending on the implementation approach [12].

The adoption of higher-order masking schemes in the physical security of block-cipher implementations has gained significant attention. This is primarily because higher-order DPA attacks become exponentially more expensive [3]. Notably, Rivain and Prouff introduced a practical $d$th-order masking scheme for AES, which incurs reasonable software implementation overhead [13]. Their approach builds upon the hardware-oriented masking scheme proposed by Ishai et al. [14]. Additionally, Coron extended the classical randomized table countermeasure against first-order attacks in their work [15]. The author presented a technique for masking lookup tables of block ciphers at any order, thereby serving as an effective countermeasure against side-channel attacks.

However, higher-order countermeasures place a significant demand on randomness during execution, which can be costly in practice, especially on resource-constrained embedded devices. To tackle this challenge, Coron et al. introduced a construction that minimizes the total amount of required randomness by utilizing multiple pseudo-random generators instead of relying solely on a True Random Number Generator (TRNG) [16].

Remarkably, real-world experiments have demonstrated that straightforward software implementations of theoretically secure runtime masking schemes are unlikely to achieve their anticipated level of security [17]. This is primarily attributed to certain sections of the code unintentionally exposing unmasked data, thus making it susceptible to side-channel analysis. In their evaluation of leakage, Beckers et al. illustrated that Test Vector Leakage Assessment (TVLA) uncovered substantial leakage, rendering nearly all the assessed implementations vulnerable to basic CPA attacks.

In hardware contexts, masking can also inadvertently lead to security vulnerabilities due to glitch-induced leakages. Standard CMOS gates, prone to glitches, may reveal power consumption that correlates with unmasked data, thereby compromising security assertions. To address these concerns, a threshold implementation derived from multi-party computation concepts has been proposed to protect digital circuits against such hardware glitches [18], [19], [20], [21]. Originally devised for hardware security, the concept of threshold implementations has also been applied to software contexts by Sasdrich et al. [22]. Their lookup table approach demonstrated a first-order secure implementation of PRESENT, indicating that properties of threshold implementations could potentially enhance software masking schemes. Subsequently, Gaspoz and Dhooghe [23] also adapted threshold implementation strategies to software for securing first-order implementations of PRESENT and Keccak variants. Their assembly implementations for RISC-V and ARM Cortex-M4 cores are platform-dependent. These software-based threshold implementations notably lack higher-order security protections and are currently limited to specific cases involving the PRESENT or Keccak algorithms. To the best of our knowledge, there are no reported instances of software implementations for more widely used block ciphers, such as AES.

To further strengthen the resistance against key leakage, a commonly employed approach by industrial practitioners is to combine higher-order masking with shuffling techniques [24]. Shuffling involves randomizing the sequence of independent operations for each encryption execution, effectively dispersing the leakage points of key-sensitive variables across $t$ different locations. This strategy mitigates the concentration of leakage at specific points, enhancing the overall security of the implementation. Provided that the sequence is shuffled uniformly at random, then the signal-to-noise ratio of the instantaneous leakage on the sensitive variable is reduced by a factor of $t$. Shuffling is easy to implement and less costly than higher-order masking especially when applied to the S-box. However, the combination of first-order masking and shuffling is not sufficiently secure against advanced power analysis [25], [26].

It’s worth mentioning that higher-order DCA [27] bears some resemblance to higher-order DPA attacks. This method was devised to retrieve secret keys from implementations that use masking and shuffling techniques by analyzing computational traces. Additionally, an algebraic DCA attack [28] has been proposed, which can break the linear masking in white-box implementations independently of the masking orders. To combat both computational and algebraic attacks, a white-box masking scheme has been introduced, combining both linear and nonlinear components [29]. However, this masking technique is a hardware-based countermeasure that is largely dependent on a run-time random number generator. In general, masking schemes are costly in terms of run time and often lead to the joint leakage. While it’s feasible to statically embed a masking technique into the
We propose a novel 8-bit linear transformation technique that achieves a balanced encoding. In contrast to the limited table diversity and ambiguity caused by using only $8 \times 8$ block invertible matrices, we present an innovative approach that incorporates non-invertible matrices within the linear transformation. This allows us to enhance the security while maintaining table diversity.

To hide the values of zeros in intermediate computations, we employ a simple nibble encoding method. Since multiplying with zeros always yields zeros, our nibble encoding replaces zeros with other nibbles in a way that preserves the balance.

We apply our proposed balanced encoding technique to an AES implementation, achieving the lowest correlation coefficients with the correct hypothetical values. To enhance security further, we employ complementary sets of lookup tables to thwart DPA-like variants targeting the lowest correlations. This AES implementation requires approximately 512KB of memory space for lookup tables and involves 1,024 table lookups.

Our experimental results underscore the effectiveness of our AES implementation in defending against various statistical analysis, including DPA-like variants, collision-based approaches, mutual information analysis, and TVLA attacks.

Importantly, our approach does not rely on masking and shuffling techniques and can be implemented without having to utilize run-time random sources in the device. This eliminates the risk of unintended unmasking during run-time and mitigates the performance overhead associated with heavyweight random number generators.

The remaining sections of the paper are organized as follows: Section II provides an overview of the fundamental concepts, including non-invasive statistical analysis, the existing method of internal encoding, and its vulnerabilities. Section III presents our proposed encoding method. In Section IV, we introduce our AES implementation that incorporates balanced encoding. We analyze its performance and cost in detail. Section V presents the experimental results, focusing on the evaluation of key protection against a range of non-invasive attacks. Section VI provides a summary of the key findings and engages in a discussion of this research.

II. Preliminaries

This section provides an overview of gray-box attacks, particularly focusing on power analysis techniques. Due to the limited space, the details can be found in the full version of this paper [31].

Power analysis is a method for extracting keys from low-cost devices like IC cards without invasive techniques. In the gray-box approach to power analysis, an attacker can choose a plaintext and access the corresponding ciphertext. They are also able to gather information on timing or power consumption during the encryption process. However, direct access to the internal workings of the computing environment, such as observing or altering memory, is not possible. Power consumption of IC cards correlates directly (or inversely) with the Hamming weight (HW) of the data being processed in the circuit. [3]. DPA and CPA are two primary techniques used in power analysis.

DCA is an alternative approach to power analysis that leverages computational traces instead of power traces. Unlike power traces, which are susceptible to electronic noise and constant components like leakage currents and transistor switching, computational traces offer a noise-free representation of the software’s read-write data or memory addresses accessed during execution. When evaluating the resistance to key leakage in cryptographic implementations, it is not always necessary to collect power traces. If the source code or execution binary is available, DCA can be employed to gather and analyze computational traces. Since computational traces are devoid of noise, they provide an efficient basis for CPA techniques. In other words, the number of traces required to recover the secret key from the implementations is reduced. To capture computational traces during the encryption process, one can utilize dynamic binary instrumentation (DBI) tools such as Intel Pin [32] and Valgrind [33]. These tools allow for the collection and analysis of computational traces. It is important to note that DBI tools can monitor binaries as they execute.
and dynamically insert new instructions into the instruction stream, rendering DCA a technique often categorized as an invasive attack.

Walsh transform can also quantify a correlation using only simple XOR and addition if the intermediate values or computational traces can be collected [4]. To evaluate the security of software cryptographic implementations, the Walsh transform can be inserted in the source code to calculate the correlation without having to use DBI tools. By doing so, scanning every sample point of the traces is not necessary. Compared to Pearson correlation coefficients, the Walsh transform consists of lightweight operations and shows the correlation as a natural number.

**Definition 1:** Let \( x = \langle x_1, \ldots, x_n \rangle \), \( \omega = \langle \omega_1, \ldots, \omega_n \rangle \) be elements of \( [0, 1]^n \) and \( x \cdot \omega = x_1\omega_1 \oplus \ldots \oplus x_n\omega_n \). Let \( f(x) \) be a Boolean function of \( n \) variables. Then the Walsh transform of the function \( f(x) \) is a real valued function over \([0, 1]^n\) that can be defined as \( W_f(\omega) = \sum_{x \in [0,1]^n} (-1)^{f(x) \cdot \omega} \).

**Definition 2:** If the Walsh transform \( W_f(x) \) of a Boolean function \( f(x_1, \ldots, x_n) \) satisfies \( W_f(\omega) = 0 \), for \( 0 \leq HW(\omega) \leq d \), it is called a balanced \( d \)-th order correlation immune function or an \( d \)-resilient function.

In Definition 1, \( W_f(\omega) \) quantifies a correlation between \( f(x) \) and \( x \cdot \omega \), where \( f(x) \) is the intermediate value and \( x \cdot \omega \) is the attacker’s hypothetical value; if \( HW(\omega) = 1 \), \( \omega \) selects a particular bit of \( x \). For each key candidate \( k^* \), for every \( \omega \in [0,1]^8 \), and for \( n \) Boolean functions \( f_i \), we can also calculate the Walsh transforms \( W_{f_i} \) and accumulate the imbalances for each key candidate using the following formula:

\[
\Delta^f_k = \sum_{\omega \in [0,1]^8} \sum_{i \in [1,n]} |W_{f_i}(\omega)|
\]

While it is possible to analyze and quantify the risk of key leakage at \( f_i \) using individual \( W_{f_i} \) values, a comprehensive understanding of key leakage can be achieved by considering the collective sum of imbalances \( \Delta^f_k \). This sum provides an overall indication of the correlation with the correct hypothetical values for each subkey candidate. In the event of key leakage, the sum of imbalances \( \Delta^f_k \) calculated using the correct subkey will exhibit a prominent spike, making it easily distinguishable from the sum of imbalances of other key candidates. In this context, the computational trace \( f(x) \) represents the noise-free intermediate value. Definition 2 outlines the criterion for a balanced first-order correlation immune function, ensuring that each bit of the intermediate values has no correlation with any bit of the hypothetical values.

From now on, we use the following notations. For 8-bit binary vectors \( X, Y, Z \), the superscripts \( H \) and \( L \) represent their upper 4 bits and lower 4 bits, respectively. Thus, \( X = X^H || X^L \). The encoding is denoted by \( \mathcal{E} \), consisting of the linear and nonlinear transformations, denoted by \( L \) and \( N \), respectively. By abuse of notation, \( N^H \) denotes the nibble encoding for the upper 4 bits whereas \( N^L \) denotes the lower 4 bits of the input. The subscripts to \( N \) are used to indicate different nibble encodings. The decoding is denoted by \( D \). Let denote two sets of \( 4 \times 4 \) binary matrices by \( \mathcal{F} \) and \( \mathcal{G} \), which are chosen under the certain conditions explained in the following section. \( f \sim \mathcal{F} \) means a random sampling from \( \mathcal{F} \). \( Idx(v) \) is defined to be a function: \( Idx(v) = \{ i \mid v_i = 1 \text{ for } i \in [1, 8] \} \), where \( v \) is an 8-bit binary vector. For example, \( \{1, 5, 6\} \leftarrow Idx([1, 0, 0, 0, 1, 1, 0, 0]) \).

**III. PROPOSED ENCODING**

Before going into depth on our balanced encoding, let’s start by explaining the imbalances in the existing 8-bit linear transformation adopted in white-box AES implementations. For \( x \in GF(2^8) \), let \( S^i(x) \) denote the SubBytes output multiplied by \( \ell \in \{1, 2, 3\} \), the elements in the MixColumns matrix. Here, \( S^i(x, y) \) refers to the \( y \)-th bit of \( S^i(x) \), where \( y \in [1, 8] \) (the MSB is the first bit). The following matrix \( S^i \) is defined as

\[
S^i = \begin{bmatrix}
S^i_{1,1} & S^i_{1,2} & \cdots & S^i_{1,256} \\
S^i_{2,1} & S^i_{2,2} & \cdots & S^i_{2,256} \\
\vdots & \vdots & \ddots & \vdots \\
S^i_{8,1} & S^i_{8,2} & \cdots & S^i_{8,256}
\end{bmatrix}
\]

An asterisk often refers to either a row or a column. For example, \( S^i_{1,s} \) refers to the \( i \)-th row, and \( S^i_{s,j} \) refers to the \( j \)-th column of \( S^i \). If \( M_{i,j} \) is an 8 \times 8 binary invertible matrix, an 8-bit linear transformation with \( S^i \) is given by \( R^i = M \cdot S^i \). Each row in \( R^i_{i,s} \) is then computed by the XOR operations between the selected rows in \( S^i \); if \( M_{i,j} = 1 \), \( S^i_{j,s} \) is XORed to compute \( R^i_{i,s} \).

By Definition 2, for random integers \( i, i' \in [1,8] \) and \( \ell, \ell' \in [1,3] \), the balanced linear transformation must satisfy the Walsh transform as follows:

\[
\sum_{j=0}^{255} (-1)^{R^i_{i',s} \cdot S^i_{j,s}} = 0.
\]

In other words, this implies \( HW(R^i_{i,s} \oplus S^i_{j,s}) = 128 \). The key-dependent distribution of \( S^i \), however, leads to \( HW(R^i_{i,s} \oplus S^i_{j,s}) = 0 \), which results in the Walsh transform value 256, with an overwhelming probability. This has been proven by the following lemma, which can be found in [34]. (Note that we have transposed matrix \( H \).)

**Lemma 1:** Assume that a 8 \times 256 binary matrix \( H \) is defined as

\[
H = \begin{bmatrix}
h_{1,1} & h_{1,2} & \cdots & h_{1,256} \\
h_{2,1} & h_{2,2} & \cdots & h_{2,256} \\
\vdots & \vdots & \ddots & \vdots \\
h_{8,1} & h_{8,2} & \cdots & h_{8,256}
\end{bmatrix},
\]

where the \( i \)-th column vector \( h_{i,s} = \langle h_{i,1}, h_{i,2}, \ldots, h_{i,8} \rangle \) is an element of \( GF(2^8) \) and \( h_{i,j} \neq h_{j,i} \) for all \( i \neq j \). Then the HW of XORs of arbitrarily chosen row vectors from \( H \) is either 0 or 128. In other words, \( HW(h_{j_1,s} \oplus h_{j_2,s} \oplus \cdots \oplus h_{j_h,s}) = 0 \)
or 128, where \( n \) is a randomly chosen positive integer and \( j_i \in \{1, 2, \ldots, 8\} \).

The proof can be simplified as follows, with more details provided in [34]. Let \( \mathcal{J} \) be a set of randomly chosen indices from \( \{1, 2, \ldots, 8\} \). We can assume without loss of generality that \( \mathcal{J} \) contains no duplicated indices and \( |\mathcal{J}| = n \leq 8 \). Define partitions of indices as

\[
\mathcal{I}_{b_1, b_2, \ldots, b_n} = \{ \ell \in \mathcal{I} | h_{j_\ell} = b_\ell \text{ for all } j_\ell \in \mathcal{J} \},
\]

where \( \mathcal{I} = \{1, 2, \ldots, 256\} \), and \( b_\ell \in \{0, 1\} \). Here all \( \mathcal{I}_{b_1, b_2, \ldots, b_n} \) are disjoint to the others and \( \cup_{b_1, b_2, \ldots, b_n} = \mathcal{I} \). For any choice of \( b_\ell \)'s, we know that

\[
|\mathcal{I}_{b_1, b_2, \ldots, b_n}| = 256/2^n = 2^{8-n}.
\]

Using the definition of the HW, it follows that HW(\( \bigoplus_{\ell \in \mathcal{J}} h_{j_\ell} \)) is summation of \( |\mathcal{I}_{b_1, b_2, \ldots, b_n}| \) where \( \bigoplus_{\ell \in \mathcal{J}} b_\ell = 1 \).

\[
\text{HW}(\bigoplus_{\ell \in \mathcal{J}} h_{j_\ell}) = \sum_{b_1, b_2, \ldots, b_n \in \{0, 1\}} \mathcal{I}_{b_1, b_2, \ldots, b_n} = 2^{8-n} = \sum_{2^{8-n}}.
\]

If \( \mathcal{J} \) is empty after removing duplicates, then the final HW is empty.

Equation (2) can be simply re-written as

\[
256 - (2 \times \text{HW}((M \cdot S^\ell)_{l, a} \oplus S^{\ell'}_{l', a})) = 256 - (2 \times \text{HW}(R^\ell_{i,j} \oplus S^{\ell'}_{i', j'})).
\]

It is important to note that for any \( \ell \) and \( \ell' \) in the range [1, 3], all row vectors of \( S^\ell \) can be represented by XORing the row vectors of \( S^{\ell'} \), and vice versa. By utilizing Lemma 1 and considering the properties of GF(2^8), it can be inferred that HW((M \cdot S^\ell)_{l, a} \oplus S^{\ell'}_{l', a}) will result in either 0 or 128, depending on the specific rows of \( S^{\ell'}_{l', a} \) that are chosen by \( M \).

Table I lists \( \mathcal{V} \) and \( \mathcal{W} \), the row index of \( S^{\ell'} \) and the sets of the row indexes of \( S^\ell \), respectively, producing the Walsh transform value of 256 for each pair of \( (\ell, \ell') \). The first row of the table means that if \( \ell = \ell' \), the Walsh transform value would be 256 only if the \( j \)-th row from \( S^\ell \) and \( S^{\ell'} \) was selected, where \( j \in \{1, 8\} \). For \( \ell, \ell' = (2, 1) \), for example, suppose that \( R^\ell \) contains a row computed by a linear combination of the 7- and 8-th rows of \( S^\ell \). This row is then identical to the 8-th bit of \( S^\ell \). In other words, this linear transformation is unable to protect \( R^\ell \) from an attacker calculating a correlation using the 8-th bit of \( S^\ell \).

Considering AddRoundKey performed before SubBytes, a subkey \( k \) will be added to the input \( x \) of \( S(\cdot) \). Here, it is easy to know that \( S(x \oplus k) \) can be expressed by a permutation of columns in \( S^\ell \) and \( S^{\ell'} \). Thus, the same row indexes listed in Table I will lead to the same results even after \( k \) is added to \( x \).

A. Basic Idea

Our encoding scheme employs a customized 8-bit linear transformation and a simple nibble encoding to protect a key-dependent intermediate byte. To maintain balance within the linear transformation, it is crucial to construct a matrix \( M \) that excludes certain row indexes, as specified in Table I.

In order to achieve this, we introduce \( \mathcal{W} \) as the set of index combinations (set) satisfying the condition:

\[
\mathcal{W} = \{J| \oplus_{\ell \in \mathcal{J}} S^{\ell}_{i,a} = S^{\ell'}_{i',a} \text{ for all } \ell, \ell' \in \{1, 3\} \},
\]

where \( i \in \{1, 8\} \). The set \( \mathcal{W} \) is constructed by gathering the index sets from Table I. To ensure the balance requirement of the linear transformation, we enforce the condition that \( Idx(M_{i,a}) \notin \mathcal{W} \), where \( i \) represents each row in matrix \( M \). This condition ensures that the selected indexes for each row of \( M \) do not coincide with the sets in \( \mathcal{W} \), thereby maintaining the desired balance within the linear transformation. By doing so, we can achieve

\[
\text{Pr}(R^\ell_{i,j} = S^{\ell'}_{i', j'}) = 1/2,
\]

which means that, from Equation (2), the observed information and the correct hypothetical value is identical with a 1/2 probability.

Furthermore, as mentioned previously, the choice of different keys only affects the order of column vectors in \( S^\ell \) and does not impact the HW of each row vector. Therefore, our encoding scheme remains independent of the key selection.

Previously, the linear transformation relied exclusively on invertible matrices to ensure its inverse transformation. However, this approach, which focused on binary invertible matrices and excluded specific row indexes, resulted in reduced table diversity.

To overcome this limitation, we have introduced a novel linear transformation that incorporates both singular and non-singular matrices. Unlike before, where decoding (i.e., inverse transformation) was not feasible when using singular matrices, our new approach successfully addresses this challenge. By incorporating both singular and non-singular matrices, we have developed a method that enables decoding even when singular matrices are utilized. This breakthrough enhances the versatility and effectiveness of the linear transformation, significantly expanding its capabilities.

Now, let’s delve into the proposed 8-bit linear transformation, which is composed of 4-bit transformations. Additionally, we utilize 4-bit nonlinear transformations, known as nibble encoding, to partially conceal the value of 0. This is necessary because multiplying any value with 0 always yields 0, which can introduce problematic correlations with correct hypothetical values. By utilizing nibble encoding, we mitigate this issue and improve the robustness and security of the linear transformation scheme.

B. Balanced Linear Transformation

1) Proposed transformation \( \mathcal{L} \): Suppose \( f \) and \( g \) are the elements randomly chosen from \( \mathcal{F} \) and \( \mathcal{G} \), respectively. For an 8-bit vector \( X = X^H||X^L \), the proposed linear transformation \( Z = L(X, f, g) \) is defined as follows:

\[
Z^H = X^H \oplus (f \cdot X^L)
\]

\[
Z^L = X^L \oplus (g \cdot Z^H),
\]

where \( \cdot \) implies multiplication. Fig. 1 illustrates a graphical representation of it. This can be simply represented by \( M \cdot X \),
TABLE I

| \(i\) | \(i'\) | \(j\) | \(W\) | \(\ell\) | \(\ell'\) | \(\nu\) | \(W\) | \(\ell\) | \(\ell'\) | \(\nu\) | \(W\) |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 1   | 2   | 1   | \{2\} | 2   | 1   | \{8\} | 3   | 1   | \{3\} | 2   | \{8\} |
| 2   | 3   | \{3\} | 2   | \{1\} | 3   | \{2\} | 4   | \{3\} | 4   | \{3\} | \{6\} |
| 3   | \{4\} | 3   | \{2\} | 3   | \{2\} | \{6\} | 4   | \{5\} | \{6\} | \{6\} | \{7\} |
| 4   | \{1, 5\} | 4   | \{3\} | 4   | \{3\} | \{5\} | 5   | \{5\} | \{6\} | \{6\} | \{7\} |
| 5   | \{1, 6\} | 5   | \{4, 8\} | 5   | \{4\} | \{5\} | 6   | \{7\} | \{6\} | \{6\} | \{7\} |
| 6   | \{7\} | 6   | \{5, 8\} | 6   | \{5\} | \{7\} | 7   | \{7\} | \{7\} | \{7\} | \{7\} |
| 7   | \{1, 8\} | 7   | \{6\} | 7   | \{6\} | \{7\} | 8   | \{7\} | \{7\} | \{7\} | \{7\} |
| 8   | \{1\} | 8   | \{7\} | 8   | \{7\} | \{7\} | \{1\} | \{2\} | \{3\} | \{4\} | \{5\} |

| \(x^i\) | \(x^{i'}\) | \(g\) | \(z^i\) |
|-----|-----|-----|-----|
| \(\oplus\) | \(\oplus\) | \(\oplus\) | \(\oplus\) |
| \(i\) | \(i\) | \(j\) | \(g\) |
| \(f\) | \(f\) | \(g\cdot f\) | \(f\cdot g\) |

Fig. 1. Balanced encoding of an 8-bit vector \(X = \text{XH}||\text{XL}\) using \(4 \times 4\) binary matrices \(f\) and \(g\).

using an \(8 \times 8\) matrix \(M\) defined as follows:

\[
\begin{bmatrix}
I_4 & f \\
G & I_4 \oplus g \cdot f
\end{bmatrix}.
\]

During the computation of \(\mathcal{L}\), if \(M_{i,j} = 1\), the \(i\)-th row of \(X\) is to be XORed. To ensure balance in the encoding, the initial step is to select random binary vectors \(f_i, g_i\) such that \(\text{Idx}(M_{i,j}) \notin W\), where \(i \in [1, 4]\).

2) Generation of \(f\): Let \(\mathcal{B}_F^i\) denote the blocklist, which represents the set of binary vectors that must not appear in the \(i\)-th row of \(f \in \mathcal{F}\). The random generation of \(f\) can be described as follows:

1) Initialize \(f\) with all elements set to 0.
2) For each row \(i\) in \(f\):
   a) Generate a 4-bit random binary vector \(b_i = [b_{i1}, b_{i2}, b_{i3}, b_{i4}]\).
   b) While \(b_i \in \mathcal{B}_F^i\), generate a new 4-bit random binary vector \(b_i\).
   c) Set the row \(i\) of \(f\) as \(b_i\).
3) Return the generated matrix \(f\).

There exist \(27,000 (= 10 \times 15 \times 15 \times 12)\) binary matrices in \(\mathcal{F}\) such that \(\mathcal{F}_{i,*} \notin \mathcal{B}_F^i\).

3) Generation of \(g\): After generating \(f\) which satisfies the condition, let \(g\) be randomly chosen from \(\mathcal{G}\). The pair \((g, f)\) is checked to ensure that \(\text{Idx}(g_{i,*} || (I_4 \oplus g \cdot f)_{i,*})\) does not belong to the set \(W\), where \(i \in [1, 4]\). Our exhaustive search has revealed that there exist a total of \(1,098,661,500 (> 2^{30})\) valid pairs of \((g, f)\) such that \(\text{Idx}(g_{i,*} || (I_4 \oplus g \cdot f)_{i,*}) \notin W\).

In other words, our linear transformation guarantees the table diversity consisting of more than \(2^{30}\) pairs of \((g, f)\) producing the Walsh transform value of 0. On average, for each \(f\), the number of row vectors at \(g_1, g_2, g_3,\) and \(g_4\) is approximately 14, 13.6, 14.2, and 14.6, respectively.

4) \(L^{-1}\) and XORs: For four 8-bit values \(X_1, X_2, X_3,\) and \(X_4\), let \(Y\) and \(Z\) stand for the followings:

\[
Y = \bigoplus_{i=1}^{4} X_i \quad \text{and} \quad Z = \bigoplus_{i=1}^{4} L(X_i, f, g).
\]
where $Z = Z^H || Z^L = \mathcal{L}(Y^H || Y^L, f, g)$. Then, the inverse of the linear transformation is accomplished by $Y = \mathcal{L}^{-1}(Z, f, g)$:

$$y^L = Z^L \oplus (g \cdot Z^H)$$

$$y^H = Z^H \oplus (f \cdot y^L)$$

$$Y = y^H || y^L.$$  

5) **Resistance to joint leakage:** One of the key features of our linear transformation $\mathcal{L}$ is its ability to maintain balance during the XOR operations of MixColumns between the encoded bytes. This property ensures resistance to joint leakage, which is a necessary condition for higher-order attacks. Therefore, our proposed implementation of AES, as described in the next section, demonstrates resilience against joint leakage, providing a robust defense against potential higher-order attacks.

Let $\ell_1, \ell_2, \ell_3, \ell_4$ represent the four coefficients of a row in the MixColumns matrix. For instance, in the first row, the coefficients are 2, 3, 1, and 1, respectively. During the MixColumns operation, each byte in the resulting state matrix is obtained by performing an XOR operation on four bytes selected from $S^1, S^2, S^3,$ and $S^4$.

To demonstrate how $\mathcal{L}$ maintains balance during these XOR computations, we introduce an $8 \times 256$ lookup table denoted as $\bar{S}$. This table is constructed as follows:

$$\bar{S}_{i,j} = S^e_{i,j} \oplus v^T$$

where $v$ is a vector used for the XOR operation. It is worth noting that $\bar{S}$ can be considered as a permutation of columns in $S^e$. Furthermore, we assume $\bar{R} = M \cdot \bar{S}$, without loss of generality. If Equation (2) is satisfied, then we have:

$$\sum_{j=0}^{255} (-1)^{R_{ij}} \bar{S}_{i,j} = 0.$$  

(4)

This equation implies that the HW($\bar{R}_{i,*} \oplus \bar{S}_{i,*}$) = 128. It provides evidence that the protection of $\mathcal{L}$ on MixColumns does not yield intermediate values that are highly correlated with the correct hypothetical value.

C. **Nibble Encoding for Hiding Zeros**

The inclusion of nibble encoding serves a specific purpose in our scheme. Prior to applying nibble encodings, the balance provided by $\mathcal{L}$ ensures that all intermediate values exhibit no problematic correlation with the correct hypothetical values. However, it is necessary to address the challenge of hiding the value of 0, which cannot be concealed through simple multiplication.

To address this issue, we employ nibble encoding, which specifically targets the hiding of 0. Each nibble encoding operation involves swapping the value 0 with a candidate value $e \in [0, 0 \times F]$, while leaving all other values unchanged. The primary objective is to identify an appropriate candidate $e$ that maintains the desired balance within the encoding process.

In summary, nibble encoding is utilized to address the requirement of concealing the value 0, which cannot be effectively hidden through standard multiplication operations.

By selectively swapping 0 with a candidate value $e$, the scheme ensures that the overall balance is maintained while effectively hiding the presence of 0 within the encoding process.

Fig. 2 shows graphical representation of two nibble encodings applied to $R^t$. In the upper and lower 4 bits, every four-bit chunk from 0 to $0 \times F$ appears exactly 16 times because of the balance by $\mathcal{L}$. Let denote two candidates to be swapped in the upper and the lower four bits by $e^H$ and $e^L$, respectively. In other words, the upper (resp. the lower) nibble encoding performs 16 swaps between 0 and $e^H$ (resp. $e^L$). Let $\mathcal{J}^H$ be a set of column indices with the upper four bits of $R^t$ equal to zero, and let $\mathcal{J}^e$ be a set of column indices with the upper four bits equal to $e$ as follows:

$$\mathcal{J}^H_0 = \{j | R^t_{1,j} || R^t_{2,j} || R^t_{3,j} || R^t_{4,j} = 0\}$$

$$\mathcal{J}^e = \{j | R^t_{1,j} || R^t_{2,j} || R^t_{3,j} || R^t_{4,j} = e\}$$

For the upper 4-bit nibble encoding $\mathcal{J}^H$, a candidate $e^H$ must satisfy the following conditions to provide the first-order balanced encoding:

$$\sum_{j \in \mathcal{J}^H_1} S^e_{1,j} = \sum_{j \in \mathcal{J}^H_2} S^e_{1,j} \text{ for all } \ell' \in \{1, 2, 3\} \text{ and } 1 \leq i \leq 8.$$  

(5)

Here, it is worthy noting that the balance must be investigated for all $\ell' \in \{1,2,3\}$ since a single matrix $M$ will be used to protect $S^1, S^2,$ and $S^3$ (see Section IV). Similarly, we define

$$\mathcal{J}^L_0 = \{j | R^t_{5,j} || R^t_{6,j} || R^t_{7,j} || R^t_{8,j} = 0\}$$

$$\mathcal{J}^L_e = \{j | R^t_{5,j} || R^t_{6,j} || R^t_{7,j} || R^t_{8,j} = e\}$$

Here, it is worthy noting that the balance must be investigated for all $\ell' \in \{1,2,3\}$ since a single matrix $M$ will be used to protect $S^1, S^2,$ and $S^3$ (see Section IV). Similarly, we define

$$\sum_{j \in \mathcal{J}^L_1} S^e_{1,j} = \sum_{j \in \mathcal{J}^L_2} S^e_{1,j} \text{ for all } \ell' \in \{1, 2, 3\} \text{ and } 1 \leq i \leq 8.$$  

(6)

A more graphical way of representing the implication is illustrated in Fig. 3. Zeros can be swapped with $e^H$ if the HW of every column in $\mathcal{J}^H_0$ and $\mathcal{J}^e$ are the same for each row of $S^t$. Also, $e^L$ can be found in the same manner using column indices belonging to $\mathcal{J}^L_0$ and $\mathcal{J}^L_e$. By using 100 different pairs of $(f, g)$, $R^t \in \{1,2,3\}$ was constructed and was then encoded by a pair of nibble encodings. We found that an average of 12,48 candidates (min: 2, max: 16) provides the balance for each nibble encoding.
cryptanalysis are not taken into account. The rest of gray- and white-box attacks such as fault injection and against DPA-like attacks which take advantage of correlation against AES [7], which is mainly composed of a series of internally encoded lookup tables. Our focus is, however, on protecting against DPA-like attacks which take advantage of correlation to the key-dependent intermediate values. In other words, the rest of gray- and white-box attacks such as fault injection and cryptanalysis are not taken into account.

IV. PROTECTED AES-128

From now on, we describe a secure design of AES with a 128-bit key protected by our balanced encoding. Overall, this is inspired by a white-box cryptographic implementation of AES [7], which is mainly composed of a series of internally encoded lookup tables. Our focus is, however, on protecting against DPA-like attacks which take advantage of correlation to the key-dependent intermediate values. In other words, the rest of gray- and white-box attacks such as fault injection and cryptanalysis are not taken into account.

A. Design

1) Rearrangement of AES: The following describes a rearrangement of AES-128 by which a series of lookup tables is generated. By shifting the initial AddRoundKey into the first round and by applying ShiftRows to the round key (except the final round key), AES-128 can be described concisely as follows:

\[
\text{state} \leftarrow \text{plaintext} \\
\text{for } r = 1 \ldots 9 \\
\quad \text{ShiftRows(state)} \\
\quad \text{AddRoundKey(state, } k^{r-1}) \\
\quad \text{SubBytes(state)} \\
\quad \text{MixColumns(state)} \\
\quad \text{ShiftRows(state)} \\
\quad \text{AddRoundKey(state, } k^9) \\
\quad \text{SubBytes(state)} \\
\quad \text{AddRoundKey(state, } k^{10})
\]

\[
\text{ciphertext} \leftarrow \text{state},
\]

where \( k^r \) is a \( 4 \times 4 \) matrix of the \( r \)-th round key, and \( \hat{k}^r \) is the result of applying ShiftRows to \( k^r \). By doing so, AddRoundKey can combine with SubBytes before multiplying each column of the MixColumns matrix for the first- to ninth-rounds.

2) Generating lookup tables up to MixColumns multiplication: We define an \( 8 \times 8 \) lookup table, denoted as \( T \)-boxes, with the following expressions:

\[
T^{i,j}_r(p) = S(p \oplus \hat{k}^{r-1})_{i,j}, \quad \text{for } i, j \in [1, 4] \text{ and } r \in [1, 9]
\]

\[
T^{i,j}_r(p) = S(p \oplus \hat{k}^{r-1})_{i,j} \oplus k^{r-1}_{i,j}, \quad \text{for } i, j \in [1, 4]
\]

where \( p \) represents a subbyte of the \( \text{state} \).

Let \([x_1 \ x_2 \ x_3 \ x_4]^T\) represent a column vector of the \( \text{state} \) after performing the lookup in the \( T \)-boxes. The subscript of each \( x \) denotes the row index of the subbyte. To precompute the multiplication of \( x_i \) with a column vector of the MixColumns matrix, we use the \( U^{r,}_{i,j} \) tables, which can be expressed as follows:

\[
U^{r,}_{i,j}(x_1) = x_1 \cdot [02 \ 01 \ 01 \ 03]^T \\
U^{r,}_{i,j}(x_2) = x_2 \cdot [03 \ 02 \ 01 \ 01]^T \\
U^{r,}_{i,j}(x_3) = x_3 \cdot [01 \ 03 \ 02 \ 01]^T \\
U^{r,}_{i,j}(x_4) = x_4 \cdot [01 \ 01 \ 03 \ 02]^T,
\]

Here, \( j \) represents the column index of the vector.

The subsequent step involves composing \( U \) and \( T \) to generate the \( UT \) tables, which map each subbyte of the plaintext to the MixColumns multiplication. It is important to note that the \( T \)-boxes, serving as the precomputation tables for SubBytes, provide the inputs for the \( U \) tables. Since the first key-dependent intermediate value, obtained by performing the encryption and looking up the \( UT \) table, is exposed, it needs to be encoded. Let us denote these values as:

\[
UT^{r,}_{i,j}(x_i) = [y_{i,1} \ y_{i,2} \ y_{i,3} \ y_{i,4}]^T.
\]

3) Encoding lookup tables: In order to apply \( L \) to the four bytes of a column vector above, four sets of \((f, g)\) are required; therefore each round uses 16 sets in total. The subscripts \( j, k \) are used to index a set pertaining to \( L \), where \( j \) is a column index and \( k \in \{1, 2, 3, 4\} \). \( y_{i,k} \) is then encoded by \( L^{r,j}_{i,k}(y_{i,k}, f_{j,k}, g_{j,k}) \). The structure of lookup tables precomputing the operations up to the MixColumns multiplication are simply depicted in Fig. 4. Note that the second round begins with the decoding of the first round’s output whereas the first round does not have to decode the plaintext. Through out this paper, we assume that the nibble encoding (blank squares in Fig. 4) is always applied to every boundary of lookup tables.

Fig. 5 illustrates a simple description of lookups from \( \text{state} \) to the first round output. For each column of \( \text{state} \), \( UT \) takes a byte and provides a 4-byte vector of the MixColumns multiplication. The intermediate values after looking up \( UT \) can be placed in a \( 4 \times 4 \times 4 \) array.

4) XOR lookup tables: The next step is to combine the encoded results of MixColumns multiplication into the round output by conducting XOR operations. Since the nibble encoding swaps zeros with unknown values, every XOR operation depicted by \( \oplus \) in Fig. 5 must be performed by looking up the XOR tables, denoted by \( T^x \). An instance of \( T^x \) is generated by using three nibble encodings to decode two 4-bit inputs and to encode a 4-bit output. Due to the distributive property of multiplication over XOR, \( L \) does not have to be decoded in
this process. The final round does not involve MixColumns, and thus $T^{10}$ is not composed with $U$. Because its outputs make a ciphertext, $T^{10}$ does not encode the output.

5) **Generating the complementary set of the balanced lookup tables:** Up to this point, we have proposed a balanced encoding scheme for the AES implementation, aimed at reducing the correlation with hypothetical intermediate values that can be exploited. However, this balanced encoding approach gives rise to two vulnerabilities. The first vulnerability stems from the lowest correlation, which increases the likelihood of the correct key being identified as the lowest-ranked key in CPA attacks. The second vulnerability arise from the bijectiveness of the encoding, making it susceptible to collision-based attacks.

To address these vulnerabilities, we propose a method that involves the random selection of one set from multiple sets, each comprising pairs of complementary lookup tables. In the subsequent discussion, we focus on the utilization of a single pair denoted as $(Q_0, Q_1)$. Let $Q_0$ represent the set of lookup tables protected by the proposed encoding, as mentioned earlier. To ensure that the correct key does not exhibit the lowest correlation or perfect collision with the correct hypothetical values, we require another set of lookup tables, denoted as $Q_1$. In $Q_1$, the input and output bits of $Q_0$ are simply flipped. As a result, $Q_1$ also provides a balanced encoding for encrypting the plaintext. In order to effectively protect the key, it is necessary to introduce a degree of imbalance. Consequently, a plaintext is encrypted using $Q_0$ with a probability of $\alpha$, while $Q_1$ is utilized with a probability of $1 - \alpha$.

Let $[b_1, b_2, \ldots, b_n]$ be a sequence of binary numbers where $b_i = 0$ or 1 such that $\sum_{i=1}^{n} b_i = n \times \alpha$. If a random number generator is available in the device, each encryption begins by picking up $i \in [1, n]$ at random and encrypts a plaintext using $Q_{b_i}$. If $\alpha = 1/2$, a binary number, say $b$, may be simply generated to choose $Q_0$. In this case, this is the only operation for which the encryption depends on the run-time random number generator. Otherwise, based on the fact that statistical analysis uses uniformly-distributed plaintexts, $i$ can be derived from plaintexts. For example, if $n$ is unknown and less than 256, an XOR sum of every subbyte of the plaintext ($mod$ $n$) can be used to choose $i$. By doing so, the encryption becomes independent of run-time random sources.

### B. Costs

Through a comparison between our single set, referred to as $Q_0$, of lookup tables and the unprotected WB-AES implementations introduced by Chow et al. [7], we observed reductions in both table size and the number of necessary table lookups achieved with $Q_0$. This enhancement stems from our choice to employ four 8-bit transformations in place of a 32-bit transformation.

Table III provides an overview of the table sizes and the corresponding number of lookups. Our balanced encoding method, as previously explained, requires a complementary set of lookup tables, where both $Q_0$ and $Q_1$ possess equal sizes, resulting in a combined table size of approximately 512KB. While there is a slight increase in size compared to Chow’s WB-AES implementation [7], which boasts a table size of around 508KB without utilizing external encoding, our approach offers a significant advantage in terms of reducing the number of required lookups. In contrast to their implementation, which demands approximately 2,032 lookups, our method achieves a notable reduction of roughly 50%. This improvement is feasible because each encryption operation only relies on just one of the table sets ($Q_0$ or $Q_1$), thereby minimizing the necessary lookup count.

In addition, Table IV presents a comparison of the elapsed time and overall memory requirement required for encrypting a single block using various AES-128 implementations. The right-most Use RNG column indicates whether a high-entropy run-time random number generator (RNG) is necessarily required. This comparison encompasses a straightforward AES implementation, Chow’s WB-AES and Lee’s WB-AES with static masking [30], along with run-time masking countermeasures such as Rivain and Prouff’s masking [13] and Coron et al.’s higher-order masking of lookup tables with common shares [35].

In particular, Rivain and Rouff’s AES masking [13] builds on the hardware-oriented scheme by Ishai et al. [14] for software implementations on general-purpose processors. Masking each S-box to secure against $d$-th order DPA requires $d$-th order secure exponentiation, generating $2d(d + 1) + 2d$ random bytes; notably, each round incorporates 16 masked S-boxes. Consequently, run-time masking schemes heavily rely on robust RNGs. In contrary, our proposed method avoids masking-based protection, obviating the need for a high-entropy RNG to secure sensitive intermediate values during cryptographic operations.

Unfortunately, there are currently no publicly available software implementations of AES threshold implementation, making a direct comparison with our implementation infeasible. It is worth noting, however, that according to [22], a lookup table approach to the software threshold implementation

---

**Table III**

| Table Size and Lookups of the Five Outer Rounds Protected by the Proposed Encoding |
|--------------------------------|
| **Size (bytes)** | **# of lookups** |
| $UT^7$ | $9 \times 4^3 \times 256 = 147,456$ | $9 \times 4^2 = 144$ |
| $T^x$ | $9 \times 4^2 \times 3 \times 2 \times 128 = 110,592$ | $9 \times 4^2 \times 3 \times 2 = 864$ |
| $T^{10}$ | $4^2 \times 256 = 4,096$ | $4^2 = 16$ |
| Total | 262,114 (approx. 256KB) | 1,024 |

---

1https://github.com/coron/htable
of PRESENT demonstrated time and space requirements comparable to the classical second-order Boolean masking using an affine transformation on the ATmega163 platform.

The experiments were carried out on a guest OS Ubuntu 16.04, running on a 3.4GHz Dual-core CPU with 8GB of RAM. Please note that the elapsed time reported for run-time masking countermeasures excludes key scheduling. To ensure a fair comparison, we did not specifically measure the elapsed times for loading the lookup tables of both WB-AES and our implementations. Our approach demonstrates a performance improvement in comparison to existing countermeasures relying on either white-box implementations or run-time masking techniques. From a practical standpoint, it’s essential to acknowledge that our implementation does impose slightly higher memory resource demands, which could be a drawback for low-cost devices. The memory requirements of our proposed method significantly contrast with the relatively lower memory demands of higher-order masking techniques. However, our implementation eliminates the requirement to generate random numbers or generate new sets of masked tables for each encryption execution. This streamlined approach greatly enhances efficiency, leading to faster encryption operations.

V. EXPERIMENTS AND RESULTS

In this section, we conduct an assessment of protection against various non-invasive statistical analysis using power traces. Building on this, we assume that the cryptographic binary cannot be extracted from the low-cost device, mitigating the risk of sophisticated invasive (white-box) attacks. Our primary focus is to evaluate the protection capabilities against well-known statistical analysis. It is worth noting that the CPA attack employing the HW model is ineffective against internally-encoded lookup tables [6]. For the sake of convenience and efficiency in our analysis, we consider scenarios in which the attacker mainly utilizes mono-bit CPA, its variants, and collision-based attacks as well as MIA and TVLA attacks. In terms of security verification, we either utilize the Walsh transform or directly collect noise-free computational traces during binary execution from memory. Subsequently, these traces undergo the aforementioned statistical analysis techniques, demonstrating our effectiveness in defending against attackers who rely on power traces collected using oscilloscopes.

A. Analysis of the UT Outputs

To illustrate the effect of the balanced encoding on the UT outputs, we present the following experimental results. First, we examined the correlation between the UT outputs and hypothetical values in the first round. These hypothetical values correspond to the outcomes of MixColumns multiplication, specifically the SubBytes outputs multiplied by 1, 2, or 3. In the subsequent analysis, we focused on $UT_{1,0}$, a segment of the UT table that maps the first subbyte of a plaintext to the MixColumns multiplication in the first round. We defined the Walsh transform $W_{t}$ as follows:

$$W_{t}(i') = \sum_{j=0}^{255} (-1)^{UT_{1,0}(j)\oplus S_{t,j}}. \tag{5}$$

In this context, $W_{t}$ quantifies the degree of correlation between the lookup values of $UT_{1,0}$ and the hypothetical SubBytes output, which is multiplied by $i' \in \{1, 2, 3\}$. Remarkably, as a consequence of the balanced encoding, all Walsh transforms yield a result of 0 for every individual bit of the correct hypothetical values. To conserve space, Fig. 6 presents a subset of the results pertaining to the hypothetical values, specifically the first bit of the SubBytes output. The comprehensive experimental findings can be accessed in [31]. In line with the details provided in Section IV, our encoding method incorporates a total of 16 sets of $(f, g)$ within each round. This meticulous arrangement ensures the protection of every subkey, validating that the random selection of $(f, g)$ preserves the desired balance within the encoding.

This balanced encoding significantly impacts the outcomes of DCA attacks that utilize the hypothetical SubBytes output. To demonstrate this effect, we collected 10,000 computational traces generated from the encryption of 10,000 random plaintexts using Valgrind. Table V presents the DCA results based on the SubBytes output. Notably, all correct subkeys ranked as the lowest among the eight attacks at least once. Table VI compares the highest correlation coefficient from all subkey candidates with the coefficient from the correct subkey. When the computational traces consist solely of the UT outputs, the DCA ranking of the correct subkey consistently remains the lowest, and the associated coefficients are notably lower than those depicted in Table VI (refer to [31] for more details). This compelling evidence underscores how the proposed encoding effectively mitigates the correlation issue inherent in the existing encoding method. These two experiments were conducted to underscore the absence of problematic correlations between the encoded lookup values and the correct values of $S_{t,e}[1,2,3]$. With $\alpha = 1/2$, the probability of encrypting a plaintext using either $Q_0$ or $Q_1$ is equal. The subsequent experiments closely resemble the previous ones, except that for each encryption, $Q_0$ and $Q_1$ were selected with a probability of 1/2. In essence, this means that the Walsh transforms were computed using the UT lookup values from both $Q_0$ and $Q_1$. Detailed results for all experiments can be found in [31]. Due to space limitations, Fig. 6 illustrates one of these experiments, where, as in previous trials, the hypothetical value represents the first bit of the SubBytes output. Notably, an increased encoding imbalance is observed, resulting in the correct subkey no longer consistently yielding the lowest correlation with the correct hypothetical values.
The encoding imbalance evident in the Walsh transform results mentioned above enhances key protection against DCA attacks seeking either the highest- or lowest-ranking subkey candidate. While Table V shows that the correct subkeys do not consistently occupy the highest or lowest ranks. Compared to Table VI, Table VIII indicates a reduction in the gap between correlation coefficients computed for the correct and wrong subkeys. For a more comprehensive understanding of the correlation coefficients based on the number of computational traces, please refer to [31].

**B. Analysis of the Round Outputs**

The analysis focused on the encoded round output of the first round to investigate the balance within the applied encoding. Let $p_1$ and $p_2$ denote the first two bytes of the plaintexts. When the remaining 14 bytes are fixed at 0, the first byte of the first round output can be expressed as follows:

$$
\delta(p_1, p_2) = S^2(p_1 \oplus k^0_{0,0}) \oplus S^3(p_2 \oplus k^0_{1,0}) \oplus c,
$$

where $c$ is a constant. For simplicity, we denote the encoding applied to $\delta$ as $\epsilon$. Then, the first subbyte of the first round output protected by the proposed encoding can be represented as $\epsilon \circ \delta(p_1, p_2)$. Assuming the attacker knows $k^0_{0,0}$, the corresponding hypothetical value is given by:

$$
\gamma(p_1, p_2) = S^2(p_1 \oplus k^0_{0,0}) \oplus S^3(p_2 \oplus k^*),
$$

where $k^*$ represents the key candidates. Using subscripts $i$ and $i'$ to denote the $i$-th and $i'$-th bits, respectively, for a fixed $p_1$ and the correct key candidate, Equation (4) suggests:

$$
255 \sum_{p_2=0}^{255} (-1)^{\epsilon \circ \delta(p_1, p_2) \oplus \gamma(p_1, p_2)} = 0.
$$

Thus, the Walsh transform $W_{\epsilon'}$ is defined as:

$$
W_{\epsilon'}(i') = \sum_{p_1=0}^{255} \sum_{p_2=0}^{255} (-1)^{\epsilon \circ \delta(p_1, p_2) \oplus \gamma(p_1, p_2)}.
$$

To assess the effect of balance in the encoded round output, we computed $W_{\epsilon'}$ for all $i, i' \in [1, 8]$. The results consistently showed zeros for the correct subkey in the Walsh transforms. For a comprehensive set of experimental results, please refer to [31]. Fig. 7a illustrates one such result, limited to the case where $i' = 1$. Only the correct subkey (0 $\times$ 55) exhibits a score of 0 for all $i \in [1, 8]$. Additionally, we collected 10,000 computational traces while encrypting plaintexts consisting of the first two random bytes ($p_1, p_2$) followed by 14 zeros. Assuming that the first subkey $k^0_{0,0}$ is known, we conducted a customized DCA attack to recover the second subkey $k^0_{1,0}$. The results are presented in Table IX. Notably, the DCA ranking of the correct subkey is relatively high compared to Table V due to a decrease in correlation coefficients computed for all key candidates. Therefore, the round output protected by the proposed encoding is
To demonstrate the disturbed collision, we collected the corresponding encoded variable in the trace also exhibits a more distinct inputs produce the same output from a function, Section II, an attacker leverages the property that when two or more distinct inputs produce the same output from a function, the corresponding encoded variable in the trace also exhibits a collision. To demonstrate the disturbed collision, we collected the following set of pairs:

\[ \mathcal{I}_v = \{(a, b) : a, b \in \{0, 1\}^8 | \rho(a, b) = v, \text{ for } v \in \{0, 1\}^8\} \]

By abuse of notation in Equation (1), we define:

\[ \Delta^{\text{coll}}_{k \in \{0, 1\}^8} = \sum_{y=0}^{255} \sum_{i,j=1}^8 \sum_{b \in \{0, 1\}} (-1)^{c_i^j \oplus v_y} \]

where \( \ell_v = |\mathcal{I}_v| \), and \( c^j_i = \epsilon \odot \delta(a^j, b^i) \) for all \((a^j, b^i) \in \mathcal{I}_v\). In cases of a perfect collision, the correct hypothetical subkey \( k^* = k \) produces the highest score \( \Delta^{\text{coll}}_{k} \), making it distinguishable from the others. However, Fig. 8a demonstrates that this distinction does not hold when \( Q_0 \) and \( Q_1 \) are randomly selected for each encryption.

Similarly, the protection against cluster analysis can be demonstrated as follows. After clustering the collected traces based on the hypothetical value \( v \), we can employ a cluster criterion function such as the sum-of-squared-error. Without loss of generality, we define:

\[ \Delta^{\text{sse}}_{k \in \{0, 1\}^8} = \sum_{y=0}^{255} \sum_{i,j=1}^8 \sum_{b \in \{0, 1\}} |c_i^j - m_i|^2 \]

where \( m_i = (\sum_{j=1}^{255} c_i^j) / \ell_v \). The optimal partition minimizes \( \Delta^{\text{sse}}_{k} \). However, Fig. 8b shows that cluster analysis is also ineffective in extracting the key.

### C. Other Considerations

We have shown that using a lookup table with balanced encoding in AES encryption significantly reduces the effectiveness of key analysis attacks through correlation-based statistical methods. Moving forward, our investigation focuses on two additional aspects. Firstly, we evaluate the capability of our proposed method to withstand MIA and TVLA attacks. Secondly, we investigate the effect of integrating our table-based implementation with existing countermeasures against fault attacks [36], [37], [38]. Specifically, our table-based implementation can be easily combined with a

---

**Table VII**

**DCA RANKING ON THE AES ENCRYPTION USING \( Q_0 \) OR \( Q_1 \) WITH A 1/2 PROBABILITY WHEN CONDUCTING MONO-BIT CPA ON THE SUBBYTES OUTPUT WITH 10,000 COMPUTATIONAL TRACES**

| TargetBut | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|-----------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|
| Key       | 48 | 51 | 41 | 199| 145| 53 | 64 | 102| 28 | 131| 67 | 15 | 245| 42 | 19 | 218|
| Key       | 175| 62 | 34 | 54 | 71 | 211| 62 | 122| 94 | 239| 213| 99 | 53 | 21 | 18 | 135|
| Key       | 4  | 65 | 57 | 66 | 76 | 230| 244| 153| 95 | 62 | 3  | 71 | 252| 219| 112| 125|
| Key       | 133| 146| 210| 214| 74 | 94 | 254| 96 | 86 | 200| 181| 232| 211| 250| 201| 191|
| Key       | 5  | 234| 124| 192| 180| 144| 186| 27 | 63 | 36 | 212| 78 | 237| 147| 238| 197|
| Key       | 18 | 57 | 136| 177| 199| 38 | 54 | 67 | 37 | 93 | 234| 93 | 38 | 52 | 100| 187|
| Key       | 4  | 66 | 10 | 70 | 88 | 38 | 177| 143| 64 | 85 | 188| 128| 161| 149| 13 | 111|
| Key       | 150| 120| 136| 27 | 110| 49 | 5  | 238| 252| 135| 212| 100| 182| 174| 46 | 127|

**Table VIII**

**HIGHEST CORRELATION OF ALL KEY CANDIDATES VS. HIGHEST CORRELATION OF THE CORRECT SUBKEYS WHEN CONDUCTING THE DCA ATTACKS ON THE UT OUTPUT OBTAINED BY \( Q_0 \) OR \( Q_1 \) WITH A 1/2 PROBABILITY**

| Subkey | 1       | 2       | 3       | 4       | 5       | 6       | 7       | 8       | 9       | 10      | 11      | 12      | 13      | 14      | 15      | 16       |
|--------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| Key    | 0.043   | 0.043   | 0.044   | 0.044   | 0.046   | 0.046   | 0.046   | 0.045   | 0.038   | 0.039   | 0.042   | 0.042   | 0.044   | 0.046   | 0.018   | 0.042   |
| Key    | 0.037   | 0.027   | 0.032   | 0.030   | 0.024   | 0.024   | 0.027   | 0.035   | 0.023   | 0.026   | 0.028   | 0.033   | 0.031   | 0.027   | 0.029   | 0.023   |

**Table IX**

**HIGHEST CORRELATION OF ALL KEY CANDIDATES VS. HIGHEST CORRELATION/RANKING OF THE CORRECT SUBKEY WHEN CONDUCTING THE CUSTOMIZED DCA ON THE FIRST ROUND OUTPUT OBTAINED BY \( Q_0 \)**

| TargetBut | 1       | 2       | 3       | 4       | 5       | 6       | 7       | 8       |
|-----------|---------|---------|---------|---------|---------|---------|---------|---------|
| Key       | 0.031   | 0.029   | 0.025   | 0.032   | 0.028   | 0.033   | 0.032   | 0.034   |
| Key       | 0.0009  | 0.0006  | 0.009   | 0.008   | 0.004   | 0.003   | 0.017   | 0.0005  |
| Key ranking | 241  | 240  | 84  | 114  | 116  | 189  | 26  | 250   |

**Table X**

**HIGHEST CORRELATION OF ALL KEY CANDIDATES VS. HIGHEST CORRELATION/RANKING OF THE CORRECT SUBKEY WHEN CONDUCTING THE CUSTOMIZED DCA ON THE FIRST ROUND OUTPUT OBTAINED BY \( Q_0 \) AND \( Q_1 \)**

| TargetBut | 1       | 2       | 3       | 4       | 5       | 6       | 7       | 8       |
|-----------|---------|---------|---------|---------|---------|---------|---------|---------|
| Key       | 0.030   | 0.027   | 0.030   | 0.034   | 0.028   | 0.030   | 0.029   | 0.030   |
| Key       | 0.009   | 0.007   | 0.011   | 0.004   | 0.01   | 0.005   | 0.004   | 0.003   |
| Key ranking | 88  | 117  | 75  | 180  | 74  | 178  | 171  | 179   |
was the SubBytes output in the first round. As depicted in
key leakage, we performed CPA attacks using the 10,000
first round. To further confirm the absence of any potential
values did not surpass the
During the TVLA analysis, it was observed that the peak
the other set was collected by encrypting random plaintexts.
which were unsuccessful in key analysis, the MIA
values, and Fig. 9 presents the attack results. Similarly to the
SubBytes’ output and the first byte of that same round’s output.
attacker’s hypothetical value derived from both the first round
during encryption processes using random plaintexts. The
pair of
in [31].
Next, we collected two sets of 10,000 computational traces,
and the final round key
into the 9-th round differential equation gives us
where 2:1:1:3 ≠ B2:37:A6:AD, indicating that differential
fault analysis is unsuccessful, as the coefficients in the 9-th
round differential equation do not match those in the Mix-
Column matrix. Based on this result, it can be inferred that
the integration with table redundancy adds to the protection against various fault injection attacks (for further details, please refer to [39]).

VI. CONCLUSION

Power analysis poses significant concerns, particularly for low-cost devices like IC cards. In many instances, cryptographic operations have been significantly slowed down to conceal key-dependent intermediate values, or memory requirements have sharply increased to enable fast and secure cryptographic operations. In this paper, we introduced an enhancement to the internal encoding of table-based AES implementations, with the aim of protecting the key hidden within the tables. In particular, we proposed a perfectly balanced encoding and demonstrated that the internally-encoded AES implementation, protected by our encoding method, consistently exhibits the lowest correlation with the correct hypothetical values. To introduce more imbalance into the encoded lookup tables, we suggested generating a complementary set of balanced tables and randomly selecting one of these table sets for encrypting plaintext. This modification guarantees the security of our AES implementation against a variety of non-invasive attacks. A notable advantage of our scheme is the memory space requirement of 512KB and the involvement of 1,024 table lookups, when compared to other proposals. Furthermore, it can be implemented without the need for run-time random number generators, rendering it highly practical for low-cost devices.

The proposed implementation provides security within the gray-box model. However, this means that if a white-box attacker successfully manipulates the execution flow to consistently select a specific set of tables, the key will still result in minimal correlation. In response to this vulnerability, our future work will be directed towards enhancing our method by incorporating a table redundancy approach, which will offer protection against both gray-box and white-box attacks.

REFERENCES

[1] P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis,” in Proc. 19th Annu. Int. Cryptol. Conf., Santa Barbara, CA, USA. Berlin, Germany: Springer, 1999, pp. 388–397, doi: 10.1007/3-540-48405-1_25.

[2] E. Brier, C. Clavier, and F. Olivier, “Correlation power analysis with a leakage model,” in Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst. Cham, Switzerland: Springer, 2004, pp. 16–29.

[3] S. Mangard, E. Oswald, and T. Popp, Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in information Security). New York, NY, USA: Springer, 2007.

[4] P. Sasdrich, A. Moradi, and T. Güneyseu, “White-box cryptography in the gray box—A hardware implementation and its side channels,” in Proc. 23rd Int. Conf. Fast Softw. Encryption (FSE), Bochum, Germany, Mar. 2016, pp. 185–203, doi: 10.1007/978-3-662-52993-5_10.

[5] J. W. Bos, C. Hubain, W. Michiels, and P. Tuexen, “Differential computation analysis: Hiding your white-box designs is not enough,” in Proc. 18th Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES). Berlin, Germany: Springer, 2015, pp. 215–236.

[6] S. Lee, T. Kim, and Y. Kang, “A masked white-box cryptographic implementation for protecting against differential computation analysis,” IEEE Trans. Inf. Forensics Security, vol. 13, no. 10, pp. 2602–2615, Oct. 2018.

[7] S. Chow, P. Eisen, H. Johnson, and P. C. V. Oorschot, “White-box cryptography and an AES implementation,” in Proc. 9th Int. Workshop Sel. Areas Cryptography (SAC). Berlin, Germany: Springer, 2002, pp. 250–270.

[8] S. Chow, P. Eisen, H. Johnson, and P. C. Van Oorschot, “A white-box DES implementation for DRM applications,” in Proc. ACM Workshop Digit. Rights Manag., Washington, DC, USA. Berlin, Germany: Springer, Nov. 2002, pp. 1–15, doi: 10.1007/3-540-44993-5_1.

[9] S. Banik, A. Bogdanov, T. Isobe, and M. Jepsen, “Analysis of software countermeasures for whitebox encryption,” IACR Trans. Symmetric Cryptol., vol. 2017, pp. 307–328, Mar. 2017. [Online]. Available: http://toc.iacr.org/index.php/ToSC/article/view/596

[10] M. Rivain and J. Wang, “Analysis and improvement of differential computation attacks against internally-encoded white-box implementations,” IACR Trans. Cryptogr. Hardw. Embed. Syst., vol. 2019, no. 2, pp. 225–255, 2019, doi: 10.13154/tches.v2019.i2.225-255.

[11] M. Zeyad, H. Maghrebi, D. Alessio, and B. Batteux, “Another look on bucketing attack to defeat white-box implementations,” in Proc. 10th Int. Workshop Constructive Side-Channel Anal. Secure Design (COSADE), Darmstadt, Germany, Apr. 2019, pp. 99–117.

[12] J. Ming, Y. Zhou, H. Li, and Q. Zhang, “A Secure and highly efficient first-order masking scheme for AES linear operations,” Cybersecurity, vol. 4, no. 1, Dec. 2021, Art. no. 14. [Online]. Available: https://api.semanticscholar.org/CorpusID:235374880

[13] M. Rivain and E. Prouff, “Provably secure higher-order masking of AES,” in Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst., S. Mangard and F.-X. Standaert, Eds. Berlin, Germany: Springer, 2010, pp. 413–427.

[14] Y. Ishai, A. Sahai, and D. Wagner, “Private circuits: Securing hardware against probing attacks,” in Advances in Cryptology—CRYPTO, D. Boneh, Ed. Berlin, Germany: Springer, 2003, pp. 463–481.

[15] J.-S. Coron, “Higher order masking of key-up tables,” in Advances in Cryptology—EUROCRYPT, P. Q. Nguyen and E. Oswald, Eds. Berlin, Heidelberg: Springer, 2014, pp. 441–458.

[16] J.-S. Coron, A. Greuet, and R. Zeitoun, “Side-channel masking with pseudo-random generator,” in Advances in Cryptology—EUROCRYPT, A. Canteaut and Y. Ishai, Eds. Cham, Switzerland: Springer, 2020, pp. 342–375.

[17] A. Beccers, L. Wouters, B. Gierlichs, B. Preneel, and I. Verbauwhede, “Provably secure software masking in the real-world,” in Proc. 13th Int. Workshop Constructive Side-Channel Anal. Secure Design (COSADE), Leuven, Belgium. Berlin, Germany: Springer, Apr. 2022, pp. 215–235, doi: 10.1007/978-3-030-99766-3_10.

[18] S. Nikova, C. Rechberger, and V. Rijmen, “Threshold implementations,” in Advances in Cryptology—CRYPTO, P. Q. Nguyen and E. Oswald, Eds. Berlin, Germany: Springer, 2006, pp. 529–545.

[19] S. Nikova, V. Rijmen, and M. Schläffer, “Secure hardware implementations of nonlinear functions in the presence of glitches,” J. Cryptol., vol. 12, no. 2, pp. 292–321, Apr. 2001, doi: 10.1007/s001450100015-010-9085-7.

[20] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen, “Higher-order threshold implementations,” in Advances in Cryptology—ASIACRYPT, P. Sarkar and T. Iwata, Eds. Berlin, Germany: Springer, 2014, pp. 326–343.
[21] T. D. Cnudde, B. Bilgin, O. Reparaz, V. Nikov, and S. Nikova, “Higher-order threshold implementation of the AES S-box,” in Proc. Int. Conf. Smart Card Res. Adv. Appl. Cham, Switzerland: Springer-Verlag, 2015, pp. 259–272, doi: 10.1007/978-3-319-31271-2_16.

[22] P. Sasdrich, R. Bock, and A. Moradi, “Threshold implementation in software: Case study of PRESENT,” in Proc. 9th Int. Workshop Constructive Side-Channel Anal. Secure Design (COSADE) in Lecture Notes in Computer Science, vol. 10815, J. Fan and B. Gierlichs, Eds. Singapore: Springer, Apr. 2018, pp. 227–244.

[23] J. Gaspoz and S. Dhooghe, “Threshold implementations in software: Micro-architectural leakages in algorithms,” IACR Trans. Cryptograph. Hardw. Embedded Syst., vol. 2023, pp. 155–179, Mar. 2023. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/10280

[24] C. Herbst, E. Oswald, and S. Mangard, “An AES smart card implementation resistant to power analysis attacks,” in RoboCup 2005: Robot Soccer World Cup IX, J. Zhou, M. Yung, and F. Bao, Eds. Berlin, Germany: Springer, 2006, pp. 239–252.

[25] S. Tillich, C. Herbst, and S. Mangard, “Protecting AES software implementations on 32-bit processors against power analysis,” in Applied Cryptography and Network Security, J. Katz and M. Yung, Eds. Berlin, Germany: Springer, 2007, pp. 141–157.

[26] S. Tillich and C. Herbst, “Attacking state-of-the-art software countermeasures—A case study for AES,” in Cryptographic Hardware and Embedded Systems—CHES, E. Oswald and P. Rohatgi, Eds. Berlin, Germany: Springer, 2008, pp. 228–243.

[27] A. Bogdanov, M. Rivain, P. S. Vejre, and J. Wang, “Higher-order DCA against standard side-channel countermeasures,” in Proc. 10th Int. Workshop Constructive Side-Channel Anal. Secure Design, Darmstadt, Germany, Apr. 2019, pp. 118–141, doi: 10.1007/978-3-030-16350-1_8.

[28] A. Biryukov and A. Udovenko, “Attacks and countermeasures for white-box designs,” in Advances in Cryptology—ASIACRYPT, T. Peyrin and S. Galbraith, Eds. Cham, Switzerland: Springer, 2018, pp. 373–402.

[29] O. Seker, T. Eisenbarth, and M. Liskiewicz, “A white-box implementation of the AES S-box,” in Proc. Int. Conf. Adv. Encrypt. Cryptograph. Standard, J. Katz and M. Yung, Eds. Berlin, Germany: Springer, Apr. 2013, pp. 108–118, doi: 10.1007/978-3-642-36362-7_9.

[30] S. Lee and N.-S. Jho, and M. Kim, “Table redundancy method for protecting AES with faulty ciphertexts only,” in Proc. Workshop Fault Diagnosis Tolerance Cryptogr., Washington, DC, USA: IEEE Computer Society, 2019, pp. 118–141, doi: 10.1007/978-3-030-16350-1_8.

[31] C. Dobrin, M. Eichlseder, T. Korak, S. Mangard, F. Mendel, and R. Primas, “SIFA: Exploiting ineffective fault inductions on symmetric cryptography,” IACR Trans. Cryptograph. Hardw. Embedded Syst., vol. 2018, no. 3, pp. 547–572, Aug. 2018, doi: 10.13154/tches.v2018.i3.547-572.

Jeong-Nyeo Kim received the M.S. and Ph.D. degrees in computer engineering from Chunjang National University, Republic of Korea, in 2000 and 2004, respectively. In 2005, she studied computer science at the University of California at Irvine, Irvine, CA, USA. Since 1988, she has been a Principal Member of the Engineering Staff with the Electronics and Telecommunications Research Institute (ETRI), where she is currently the Project Leader of the Network System Security Research Section. Her research interests include the IoT security, mobile security, secure OS, system security, and network security.