IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2024, No. 3, pp. 634–670. DOI:10.46586/tches.v2024.i3.634-670

Unboxing ARX-Based White-Box Ciphers: Chosen-Plaintext Computation Analysis and Its Applications

Yufeng Tang[1], Zheng Gong[1](✉), Liangju Zhao[1], Di Li[1] and Zhe Liu[2]

[1] School of Computer Science, South China Normal University, Guangzhou, China
cis.gong@gmail.com
[2] Zhejiang Lab, Hangzhou, China

Abstract. It has been proven that the white-box ciphers with small encodings will be vulnerable to algebraic and computation attacks. By leveraging the large encodings, the *self-equivalence* and *implicit* implementations are proposed for ARX-based white-box ciphers. Unfortunately, these two types of white-box implementations are proven to be insecure under the algebraic attack. Different from algebraic attacks, computation analysis can extract the secret key from the memory access traces without software reverse engineering. It is still an open problem whether the self-equivalence and implicit implementations can resist the computation analysis.

In this paper, we analyze the encoded structure of the self-equivalence/implicit white-box ARX ciphers and discuss its resistance to the computation analysis, such as *differential computation analysis* (DCA) and *algebraic degree computation analysis* (ADCA). The results reveal that the large input, encoding, and round key can practically mitigate DCA and ADCA. To deal with the large space, we introduce a new method which is named *chosen-plaintext computation analysis* (CP-CA). Based on a partial key guess and deliberately chosen intermediate value, CP-CA constructs a reverse function to calculate a set of plaintexts. With the obtained plaintexts, the large affine and non-linear encodings will be reduced to a small space. Subsequently, CP-CA mounts the computation analysis on the traces to recover the secret key. Following CP-CA, we propose a CP-DCA attack and reformulate ADCA as *chosen-plaintext linear encoding analysis* (CP-LEA). The experimental results indicate that the self-equivalence white-box SPECK32/48/64/96/128 and implicit white-box SPECK32/64 implementations are vulnerable to CP-DCA and CP-LEA attacks.

Keywords: White-box implementation · Self-equivalence encoding · Implicit function · Differential computation analysis · Algebraic degree computation analysis

1 Introduction

The *white-box* attack context assumes that an attacker has full control over the execution environment. The sensitive values such as the secret key can be extracted by memory dumps. The adversary can also set a breakpoint to the algorithm and manipulate the executed primitive by reverse engineering. The seminal works on *white-box cryptography* [CEJvO02a, CEJvO02b] were introduced by Chow *et al.* in 2002. They proposed the first white-box AES and DES implementations to prevent the *key extraction* attack in the white-box context. Their method of white-boxing a cipher is called the CEJO framework. The main idea of CEJO is to convert the subround functions into a series of look-up tables (LUTs) with embedded subkeys. Subsequently, each LUT is composed with random *encodings* to obfuscate the key information. The applied encodings can be divided into two categories,

Received: 2024-01-15 Accepted: 2024-03-15 Published: 2024-07-18

namely *internal* and *external* encodings. Internal encodings can be canceled pairwise between two successive LUTs. Differently, external encodings are applied to the input and output of the cryptographic algorithm. It modifies the primitive by encoding the first-round input and the final-round output. Following CEJO, many white-box implementations [LN05, BCD06, Kar10, BCH16] have been proposed. Unfortunately, based on the properties of collision and affine equivalence, both the encodings and the secret key can be extracted from the CEJO framework and its variants [BGE04, WMGP07, MWP10, LRM⁺13, DFLM18].

At CHES 2016, *differential computation analysis* (DCA) [BHMT16] was proposed as the software counterpart of *differential power analysis* (DPA) [KJJ99] to the white-box context. DCA requires that the white-box implementation is constructed without the input or output external encoding. It mounts a statistic analysis on the *computation trace* which consists of the runtime computed values of the cryptographic algorithm. The principle behind DCA is that one can obtain a high correlation between the traces and the key-dependent sensitive values. A DCA adversary only needs to analyze the accessed memory during the execution of the implementation without the knowledge of encoding details. It has been demonstrated that many publicly available white-box solutions are vulnerable to DCA. Moreover, since DCA can be mounted automatically, it is an effective attack against the white-box challenges [BT20]. Hereafter, many other computation analyses have been proposed with different attack methods, such as *spectral analysis* [SMG16] and *mutual information analysis* [RW19]. Bock *et al.* [BBMT18, BBB⁺19] exhibited the weakness of 4-bit non-linear encoding and demonstrated the ineffectiveness of the internal encodings against DCA. Carlet *et al.* [CGM21] introduced a new spectral analysis to defeat the 8-bit non-linear encoding. Recently, Tang *et al.* [TGLZ23] compared the capabilities of different published methods and proposed a generic *algebraic degree computation analysis* (ADCA) by exploiting the degree of encodings. Compared with various analyses, ADCA can defeat most cases of encodings with the lowest time complexity. Based on these observations, CEJO is vulnerable to both algebraic attack and computation analysis.

Different from CEJO with small encodings, a *self-equivalence* framework [MS16] was proposed to construct a white-box implementation with large encodings. The self-equivalence is a pair of affine functions (A, B) such that $S = B \circ S \circ A$ for an Sbox layer S. It combines the secret key with the full-round linear layer and applies large self-equivalence encoding to hide the key information. Ranea and Preneel [RP20] analyzed the security of substitution-permutation network (SPN) ciphers with self-equivalence encodings. The substitution layer of an SPN cipher is a concatenation of small and cryptographically strong Sboxes. They proved that the self-equivalences of a cryptographically strong Sbox have a diagonal shape. Moreover, a small Sbox only has a few pairs of self-equivalences, such as 2040 pairs of AES [BCBP03]. Thus, the self-equivalence encodings are inadequate to protect the SPN ciphers. Motivated by this work, Vandersmissen *et al.* [VRP22] applied the self-equivalence encodings to protect the add-rotate-xor (ARX) ciphers and proposed an instantiated self-equivalence SPECK [BSS⁺13] (SE-SPECK). An ARX cipher does not rely on a strong Sbox to provide the nonlinearity. Its non-linear layer contains a large modular addition which is not a concatenation of small Sboxes. Moreover, compared with an AES Sbox, an n-bit modular addition has a larger number of self-equivalences, which is exponential in n. However, the self-equivalence of SPECK has a sparse matrix. Vandersmissen *et al.* [VRP22] introduced an algebraic attack to recover the self-equivalence encodings and extract the secret key of SE-SPECK.

To hide the sparse structure of self-equivalence, Ranea *et al.* [RVP22] proposed an *implicit* framework for ARX ciphers and implemented a SPECK instance (IF-SPECK). It applies $2n$-bit large affine or affine-quadratic self-equivalence encoding to obfuscate the key information. Furthermore, the implicit implementation represents each round function by a low-degree implicit function without the exposure of the encoding structure. Each implicit round function is implemented as binary multivariate polynomials. It solves an affine system to obtain the round output based on a round input. Ranea *et al.* [RVP22] stated that the implicit implementation can resist all known white-box algebraic attacks when using a quadratic input encoding or a large non-linear layer. However, Biryukov *et al.* [BLU23] proposed an algebraic attack on IF-SPECK. The results demonstrated that when input or output external encoding is omitted, *linear decoding analysis* (LDA) [BU18, GPRW20] can break the implementation with the time complexities $\mathcal{O}(n^3)$ for affine encoding and $\mathcal{O}(n^6)$ for quadratic encoding.

Motivation. To the best of our knowledge, there is no published result on the computation analysis against self-equivalence and implicit function frameworks, especially the DCA attack which computes the correlations between the sensitive values and the collected traces. Although the cryptanalysis on IF-SPECK without the external encoding is similar to the case of computation analysis, LDA requires access to the full state of the encoded value to ensure the decoding system contains all the target bits. This implies that an LDA attacker needs to locate a large window in the traces. Diversely, for DCA-style computation analysis, it is sufficient to focus on some single intermediate bits in the trace at a time without the full state of the encoded value. Therefore, the resistances of SE-SPECK and IF-SPECK in the DCA context have not been analyzed.

Our Contribution. This paper focuses on solving the open problems to evaluate the security of the self-equivalence and implicit ARX implementations against the computation analysis without either the input or the output external encoding. For a better illustration of our attack, we mainly focus on the detailed descriptions of the white-box SPECK implementation with a trivial input external encoding. Our contribution consists of the following three parts.

1. **Computation analysis against the encoded ARX structure.** We first refine an encoded target function which represents the combination of the first few rounds of SE-SPECK and IF-SPECK. Since DCA is the original analysis that computes the correlation while ADCA is the latest one that focuses on the algebraic degree, we consider the attack instances of DCA and ADCA against SE-SPECK and IF-SPECK. The theoretical analysis indicates that DCA and ADCA cannot be mounted practically because of the large dimensions of input, key guess, and encoding.

2. **A new chosen-plaintext computation analysis.** To deal with the large encoding phase, we introduce the concept of a reverse function. It is constructed by some reverse operations of the cryptographic algorithm and needs to be initialized with a key guess. The introduction of the reverse function helps to reduce the large encoding into a small one. It can bypass the high time complexity of the encoding, input, and key enumerations over the full space. We also propose a new *chosen-plaintext computation analysis* (CP-CA) with three attack phases, namely partial key guess, partial key verification, and round key verification. By fixing some parts of the inputs and guessing the partial key, CP-CA can distinguish the correct key by the computation analysis on the traces.

3. **Practical attacks on ARX-based white-box ciphers.** For concrete instances, we construct a CP-DCA attack and reformulate ADCA as a new *chosen-plaintext linear encoding analysis* (CP-LEA) following the CP-CA model. CP-DCA computes the linear enumerations for the reduced encodings while CP-LEA constructs a linear system to detect the correct key. Although CP-LEA seems like an algebraic attack instead of a correlation-based one, it still focuses on analyzing the traces as the computation analysis. Moreover, CP-DCA and CP-LEA are extended as higher-degree attacks to break the non-linear encodings. For block size $2n$, Table 1 compares

the time complexities of LDA, DCA, and CP-CA to attack the ARX-based white-box ciphers. To counteract affine encodings, DCA requires an exhaustive search on 2^{2n} linear encodings and 2^n key candidates. The symbol t represents the length of computation traces. CP-CA generates the reverse functions based on $|\mathcal{K}|$ key candidates and constructs N n_b-bit chosen inputs. To break the quadratic encoding, CP-CA applies a degree-2 extension to the chosen inputs. The symbol q denotes the number of the extended higher-degree bits and $p = q + 1$. Particularly, DCA cannot defeat the large non-linear encoding. To validate the CP-CA, we mount the practical CP-DCA and CP-LEA attacks on SE-SPECK-32/48/64/96/128 and IF-SPECK-32/64. The attack results (refer to Table 5 in Section 6.2) reveal that both SE-SPECK and IF-SPECK are vulnerable to the CP-CA attacks. Moreover, the practical attack results are also demonstrated on the self-equivalence and implicit white-box CRAX [BBdS+20] implementations (refer to Table 6 in Section 6.5). The source code of our experiments is publicly available [1].

Table 1: The theoretical time complexities of LDA, DCA, and CP-CA on the round key recovery of ARX-based white-box ciphers with affine and quadratic encodings.

Encoding	LDA ([BLU23])	DCA ([BBB+19])	CP-CA (Section 4)	
			CP-DCA (Section 5.1)	CP-LEA (Section 5.2)
Affine	$\mathcal{O}(t^3)$	$\mathcal{O}(t \cdot N \cdot 2^{3n})$	$\mathcal{O}(n/n_b \cdot \lvert\mathcal{K}\rvert \cdot 2^{n_b} \cdot t \cdot N)$	$\mathcal{O}(n/n_b \cdot \lvert\mathcal{K}\rvert \cdot t \cdot N \cdot (n_b + 1))$
Quadratic	$\mathcal{O}(t^6)$	-	$\mathcal{O}(n/n_b \cdot \lvert\mathcal{K}\rvert \cdot 2^q \cdot t \cdot N)$	$\mathcal{O}(n/n_b \cdot \lvert\mathcal{K}\rvert \cdot t \cdot N \cdot p)$

Organization. Section 2 introduces the preliminaries of ARX-based white-box ciphers and computation analysis. Section 3 analyzes the encoded structure of SE-SPECK and IF-SPECK. Its resistance against DCA and ADCA is also discussed. The CP-CA method is proposed in Section 4. Section 5 provides a detailed description of two attack instances of CP-DCA and CP-LEA. Section 6 illustrates the experimental results and the comparisons of CP-CA. Section 7 concludes this paper.

2 Preliminaries

2.1 Notions and Notations

Let $\mathbb{F}_2^n$ be the vector space of an n-bit value. The operator $\oplus$ denotes the addition in $\mathbb{F}_2$, which is also called an exclusive-or (`XOR`) operation. A right circular shift of x by α bits is denoted by $x \ggg \alpha$ while a left circular shift of x by β bits is denoted by $x \lll \beta$. A modular addition $x \boxplus y$ between two numbers $x \in \mathbb{F}_2^n$ and $y \in \mathbb{F}_2^n$ is defined as an addition modulo 2^n. Its inverse operation, namely modular subtraction, is represented by $x \boxminus y$. A Boolean function of n variables is a mapping $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$. A function $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ is called (n, m)-bit function and refers to an n-bit function if $n = m$. An identity function is represented by F_{id}. The *Hamming weight* (`HW`) of a vector is the number of its nonzero coordinates. The concatenation of vectors a and b is represented as $a \parallel b$. The *algebraic normal form* (ANF) is to represent a Boolean function as a polynomial on n variables, with coefficients over $\mathbb{F}_2$. Every variable in each monomial has a degree 0 or 1. The *degree* of a Boolean function is the maximum degree of the monomials in its ANF. The degree of an (n, m)-bit function is the maximum degree of its m coordinate functions. A function F is affine if its degree is 1. An n-bit affine function F can be represented by a matrix

[1] https://github.com/scnucrypto/CP-CA

multiplication and a vector addition, such that $F = Ax \oplus a$ where A is an $n \times n$ matrix and a is a non-zero n-bit vector. An affine function F is linear if $F(0) = 0$ and it satisfies $F(x) = Ax$ with the constant vector a is 0. The correlation (Pearson correlation coefficient) between two n-bit vectors u and v is defined as

$$\texttt{Cor}(u, v) = \frac{n_{11}n_{00} - n_{01}n_{10}}{\sqrt{(n_{00} + n_{01})(n_{00} + n_{10})(n_{11} + n_{01})(n_{11} + n_{10})}},$$

where n_{ij} represents the number of positions where u equals to i and v equals to j. If the denominator equals zero, the correlation is set to zero. The correlation satisfies $-1 \leq \texttt{Cor}(u, v) \leq 1$. If $|\texttt{Cor}(u, v)| = 1$, u and v are linearly correlated. If $\texttt{Cor}(u, v) = 0$, u and v are linearly independent.

2.2 ARX Ciphers and SPECK

ARX ciphers consist of three arithmetic operations which are modular addition, bitwise rotation, and XOR. The non-linear layer of ARX ciphers contains the modular addition without the Sbox. ARX is a flexible and lightweight structure for designing cryptographic algorithms. SPECK [BSS+13] is an ARX block cipher with a Feistel-like structure. It has a block size of $2n$ bits where n is the word size for $n = 16/24/32/48/64$. Thus, it has 5 variants, namely SPECK32/48/64/96/128. The size of the master key is $m \cdot n$ where $m = 2/3/4$. The number of encryption rounds n_r ranges from 22 to 34 for different versions of SPECK. Let $x_0^{(r)}$ and $x_1^{(r)}$ denote the half n-bit inputs to r-th round. The n-bit round key in the r-th round is denoted by $k^{(r)}$. The round function for SPECK is computed as follows. The rotation constants are specified as $\alpha = 7$, $\beta = 2$ for SPECK32 and $\alpha = 8$, $\beta = 3$ for other variants.

$$\begin{aligned} x_0^{(r+1)} &= \left((x_0^{(r)} \ggg \alpha) \boxplus x_1^{(r)}\right) \oplus k^{(r)} \\ x_1^{(r+1)} &= (x_1^{(r)} \lll \beta) \oplus x_0^{(r+1)} \end{aligned}$$

2.3 White-Box SPECK Implementations

Following the CEJO framework, white-box implementations apply the encodings to protect the inputs and outputs of key-dependent round functions.

Definition 1 (Encoding [RP20])**.** Let F be an (n, m)-bit function. Let (I, O) be a pair of n-bit and m-bit permutations, respectively. An encoded function of F is defined as $\overline{F} = O \circ F \circ I$, where I and O are called the input and output encoding, respectively.

Definition 2 (Encoded Encryption Function [RP20, RVP22])**.** Let $E_k = E^{(n_r)} \circ \cdots \circ E^{(1)}$ be the encryption function of an iterated n-bit cipher with fixed key k. A white-box implementation $\overline{E_k}$ is an encoded E_k composed of encoded round functions, that is,

$$\overline{E_k} = \overline{E^{(n_r)}} \circ \cdots \circ \overline{E^{(1)}} = (O^{(n_r)} \circ E^{(n_r)} \circ I^{(n_r)}) \circ \cdots \circ (O^{(1)} \circ E^{(1)} \circ I^{(1)}),$$

where the n-bit round encodings $(I^{(r)}, O^{(r)})$ satisfying $I^{(r+1)} \cdot O^{(r)} = F_{id}$ for $r = 1, 2, \cdots, n_r - 1$.

Because of the pairwise canceling encodings between the successive rounds, the encoded encryption can also be represented as $\overline{E_k} = O^{(n_r)} \circ E_k \circ I^{(1)}$. The encodings $(I^{(1)}, O^{(n_r)})$ are the external encodings. In the following analysis, we suppose that the white-box implementation is constructed without the input external encoding. Thus, $I^{(1)} = F_{id}$ and the inputs of the white-box implementation are equivalent to the inputs of the standard cipher. By leveraging the self-equivalence of the non-linear layer, two types of white-box implementations are proposed for white-boxing ARX ciphers, which are self-equivalence and implicit frameworks.

Self-Equivalence Encoding. SE-SPECK rearranges the components of the round function E_k into a SPN structure. Each round function consists of a substitution layer S and a following affine layer AL, except for the first round which only contains an affine layer, such that

$$E_k = (AL^{(n_r)} \circ S) \circ \cdots \circ (AL^{(1)} \circ S) \circ AL^{(0)}.$$

The constructions of the affine and substitution layers are described as follows.

$$\begin{aligned} S(x_0, x_1) &= (x_0 \boxplus x_1, x_1) \\ AL^{(0)}(x_0, x_1) &= (x_0 \ggg \alpha, x_1) \\ AL^{(r)}_{1 \le r \le n_r - 1}(x_0, x_1) &= \Big((x_0 \oplus k^{(r)}) \ggg \alpha, (x_1 \lll \beta) \oplus (x_0 \oplus k^{(r)})\Big) \\ AL^{(n_r)}(x_0, x_1) &= \Big(x_0 \oplus k^{(n_r)}, (x_1 \lll \beta) \oplus (x_0 \oplus k^{(r)})\Big) \end{aligned}$$

Definition 3 (Affine (Linear) Self-Equivalence [RP20])**.** Let F be an n-bit function. An affine (resp. linear) self-equivalence of F is a pair of affine (resp. linear) functions (A, B) such that $F = B \circ F \circ A$.

As depicted in Figure 1, SE-SPECK applies the self-equivalence encodings of S to each of the key-dependent affine layers. The encoded $\overline{AL^{(r)}}$ is defined as

$$\overline{AL^{(r)}}_{2 \le r \le n_r - 1} = A^{(r)} \circ AL^{(r)} \circ B^{(r-1)},$$

where $A^{(r)}$ and $B^{(r)}$ are affine (linear) self-equivalence encodings satisfying $B^{(r)} \circ S \circ A^{(r)} = S$. Since the first-round affine layer $AL^{(0)}$ does not contain any key material, it is not protected using self-equivalence encoding. Moreover, owing to the absence of external encodings, the affine layers $\overline{AL^1}$ and $\overline{AL^{n_r}}$ are constructed as

$$\overline{AL^{(1)}} = A^{(1)} \circ AL^{(1)}, \ \overline{AL^{(n_r)}} = AL^{(n_r)} \circ B^{(n_r - 1)}.$$

The substitution layer S with its self-equivalence can be reduced to S. Thus, the introduced self-equivalence encodings can be canceled by composing the round functions. The resulting self-equivalence implementation $\overline{E_k}$ is computed as

$$\begin{aligned} \overline{E_k} &= (\overline{AL^{(n_r)}} \circ S) \circ \cdots \circ (\overline{AL^{(1)}} \circ S) \circ AL^{(0)} \\ &= AL^{(n_r)} \circ (B^{(n_r-1)} \circ S \circ A^{(n_r-1)}) \circ \cdots \circ (B^{(1)} \circ S \circ A^{(1)}) \circ AL^{(1)} \circ S \circ AL^{(0)} \\ &= AL^{(n_r)} \circ S \circ \cdots \circ S \circ AL^{(1)} \circ S \circ AL^{(0)} = E_k. \end{aligned}$$

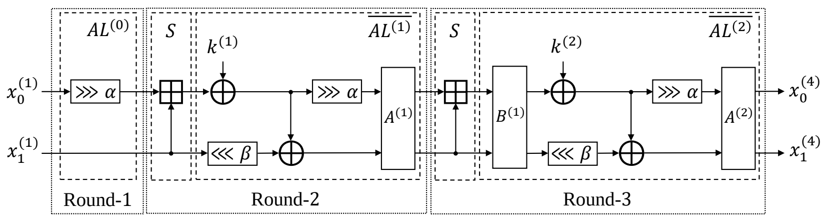

Figure 1: First three encryption rounds of SE-SPECK.

Implicit Function. IF-SPECK represents the round function by a quasilinear implicit function and protects it with large self-equivalence encodings. An implicit function can be implemented as multivariate binary polynomials.

Definition 4 (Implicit Function [RVP22])**.** Let F be an n-bit function. An implicit function of F is a $(2n, m)$-bit function P satisfying $P(x, y) = 0 \Leftrightarrow y = F(x)$.

Definition 5 (Quasilinear Function [RVP22])**.** A $(2n, m)$-bit implicit function P is quasilinear if for all $x \in \mathbb{F}_2^n$ the (n, m)-bit function $y \mapsto P(x, y)$ is affine.

We assume that the r-th round function $E^{(r)}$ of an ARX cipher consists of a non-linear layer S and a linear layer $AL^{(r)}$ with a round key. It satisfies $E^{(r)} = S \circ AL^{(r)}$. Let T denote a quasilinear implicit function of S such that $T(x, y) = 0 \Leftrightarrow y = S(x)$. Let $I^{(r)}$ and $O^{(r)}$ represent the input and output encodings, respectively. The encoding $O^{(r)}$ needs to be affine for the quasilinear property while there is no restriction of the encoding $I^{(r)}$. Let $(A_S^{(r)}, B_S^{(r)})$ be the self-equivalence of S, which satisfies $S = B_S^{(r)} \circ S \circ A_S^{(r)}$. Let $V^{(r)}$ denote a random linear encoding satisfying $V^{(r)}(0) = 0$. We can compute a quasilinear implicit function $P^{(r)}$ of the round function $E^{(r)}$ as

$$P^{(r)} = V^{(r)} \circ T \circ (A_S^{(r)}, (B_S^{(r)})^{-1}) \circ (AL^{(r)}, F_{id}) \circ (I^{(r)}, (O^{(r)})^{-1}).$$

Based on the property of implicit function, we have

$$\begin{aligned} P^{(r)}(x, y) = 0 &\Leftrightarrow T \circ (A_S^{(r)} \circ AL^{(r)} \circ I^{(r)}(x), (B_S^{(r)})^{-1} \circ F_{id} \circ (O^{(r)})^{-1}(y)) = 0 \\ &\Leftrightarrow (B_S^{(r)})^{-1} \circ (O^{(r)})^{-1}(y) = S \circ A_S^{(r)} \circ AL^{(r)} \circ I^{(r)}(x) \\ &\Leftrightarrow y = O^{(r)} \circ B_S^{(r)} \circ S \circ A_S^{(r)} \circ AL^{(r)} \circ I^{(r)}(x) \\ &\Leftrightarrow y = O^{(r)} \circ S \circ AL^{(r)} \circ I^{(r)}(x) = O^{(r)} \circ E^{(r)} \circ I^{(r)}(x). \end{aligned}$$

For considering non-linear self-equivalence, if A is affine and B is quadratic, (A, B) is called an *affine-quadratic self-equivalence*. Let C denote a random affine encoding. Let $(A^{(r)}, B^{(r)})$ be an affine or affine-quadratic self-equivalence of $E^{(r)}$, the round encodings are defined as

$$\begin{aligned} (I^{(r)}, O^{(r)}) &= \left(A^{(r)} \circ B^{(r-1)} \circ (C^{(r)})^{-1}, C^{(r+1)}\right) \\ &= \left(B^{(r-1)} \circ (C^{(r)})^{-1}, C^{(r+1)} \circ (B^{(r)})^{-1}\right). \end{aligned}$$

For two consecutive rounds, the input and output encodings cancel out each other as follows.

$$\begin{aligned} &\left(O^{(r+1)} \circ E^{(r+1)} \circ I^{(r+1)}\right) \circ \left(O^{(r)} \circ E^{(r)} \circ I^{(r)}\right) \\ =& \left(C^{(r+2)} \circ E^{(r+1)} \circ A^{(r+1)} \circ B^{(r)} \circ (C^{(r+1)})^{-1}\right) \circ \\ &\left(C^{(r+1)} \circ E^{(r)} \circ A^{(r)} \circ B^{(r-1)} \circ (C^{(r)})^{-1}\right) \\ =& C^{(r+2)} \circ E^{(r+1)} \circ A^{(r+1)} \circ B^{(r)} \circ E^{(r)} \circ A^{(r)} \circ B^{(r-1)} \circ (C^{(r)})^{-1} \\ =& C^{(r+2)} \circ E^{(r+1)} \circ A^{(r+1)} \circ E^{(r)} \circ B^{(r-1)} \circ (C^{(r)})^{-1} \\ =& C^{(r+2)} \circ (B^{(r+1)})^{-1} \circ E^{(r+1)} \circ E^{(r)} \circ B^{(r-1)} \circ (C^{(r)})^{-1} \end{aligned}$$

If B is quadratic, the input and output encodings are non-linear. If B is affine, the input and output encodings are affine. We note that the first round does not contain any key information. Thus, it is constructed without encodings. Since the absence of the external input encoding, the second round function is constructed without the input encoding. Figure 2 depicts the explicit functions of the first two rounds of IF-SPECK.

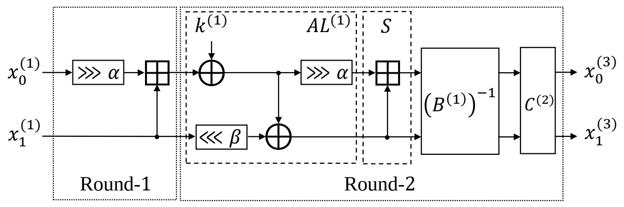

Figure 2: First two encryption rounds of IF-SPECK.

2.4 Computation Analysis

Attack Model. Following the passive attack model [RW19], computation analysis assumes that the attacker can invoke the white-box implementations many times with arbitrary plaintexts. It analyzes the intermediate values during the execution of the cryptographic algorithm. The computation traces are recorded with the help of *dynamic binary instrumentation* (DBI) tools. Therefore, computation analysis can be regarded as a gray-box technique against white-box implementations, without the knowledge of implementation details and reverse engineering efforts.

Definition 6 (Computation Trace)**.** A computation trace $\boldsymbol{v} \in \mathbb{F}_2^t$ consists of t samples, such that $\boldsymbol{v} = \{v_1, v_2, \cdots, v_t\}$. Each sample $v_i \in \mathbb{F}_2$ for $1 \le i \le t$ corresponds to an intermediate computed variable of the cryptographic algorithm.

Definition 7 (Selection Function)**.** Let $\mathcal{K}$ and $\mathcal{X}$ denote the vector spaces of key and input, respectively. A selection function $\varphi_k(x)$ computes a sensitive value that depends on a key guess $k \in \mathcal{K}$ and an input $x \in \mathcal{X}$ to the cryptographic algorithm.

Definition 8 (Distinguisher)**.** For $1 \le i \le N$, a distinguisher δ_k is a function $\mathtt{D}$ which maps a selection function with N inputs $x^{(i)}$, and N corresponding computation traces $\boldsymbol{v}^{(i)}$ to a score vector, such that

$$(\delta_k)_{k\in\mathcal{K}} = \mathtt{D}\left(\left(\varphi_k(x^{(1)}), \varphi_k(x^{(2)}) \cdots, \varphi_k(x^{(N)})\right), (\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)} \cdots, \boldsymbol{v}^{(N)})\right).$$

The proposed computation analysis methods apply different distinguishers to recover the secret key by analyzing the traces and the inputs. Thus, different methods have various capabilities to defeat the encodings for recovering the secret key. To the best of our knowledge, ADCA is the latest computation analysis against the internal encodings. Although DCA is less effective than ADCA, it is the first analysis based on correlation computation. The analysis of DCA can help to discuss the resistance of the encoded ARX structure against the correlation-based attack. Thus, this paper mainly focuses on the DCA and ADCA attacks.

DCA. DCA first collects the computation traces by invoking the white-box implementation with random inputs. Subsequently, it computes the correlations between the hypothetical values and the collected computation traces. Each hypothetical value is the output of a selection function computed by a key guess. Such a DCA distinguisher can be described as follows.

$$\delta_k^{\mathtt{DCA}} = \mathtt{arg\ max}\,|\mathtt{Cor}((\varphi_k(x))_i, \boldsymbol{v}_j)|$$

To analyze a white-box implementation without the external input encoding, a selection function is the first-round Sbox such that $\varphi_k(x) = S(x \oplus k)$ for a key guess $k \in \mathcal{K}$. DCA calculates the correlations between the i-th coordinate output of the selection function $(\varphi_k(x))_i$ and the j-th sample of traces $\boldsymbol{v}_j$. The highest computed correlation is assigned as the rank of the corresponding key guess. The key guess with the highest correlation is the most likely correct one.

ADCA. ADCA is a higher-degree computation analysis that was proposed for the CEJO structure. The attack procedure of ADCA consists of two phases, namely the chosen-input phase and the degree-computation phase. The target function can be defined as $O_{k^*}(x) = EC \circ S \circ \oplus_{k^*}$ where EC denotes an encoding and k^* is the secret key. In the chosen-input phase, ADCA collects the computation traces by choosing the inputs $x' = S^{-1}(x) \oplus k$ for a key guess $k \in \mathcal{K}$. With the chosen inputs x', the target function can be transformed as $A_k(x) = O_{k^*}(x') = EC \circ S \circ \oplus_{k^*} \circ \oplus_k \circ S^{-1}$. For a correct key guess $k = k^*$, the target function A_{k^*} is transformed into the encoding function EC. Thus, the degree of A_{k^*} is equal to the one of EC. For an incorrect key guess $k \neq k^*$, the target function $A_k = EC \circ S \circ c \circ S^{-1}$ where c is a non-zero vector. Because of the high degree of an Sbox (degree $n-1$ for an n-bit Sbox), the degree of an n-bit A_k is most likely $n-1$. When the degree of the encoding EC is less than $n-1$, A_{k^*} can be distinguished from the random functions A_k for its lower degree. Thus, the correct key k^* can be recovered. In the degree-computation phase, ADCA detects the degrees of the mappings $(f_k)_i : \mathbb{F}_2^n \mapsto \mathbb{F}_2 : (f_k)_i(x) = v_i$ from the inputs $x \in \mathbb{F}_2^n$ to each sample of traces $v_i \in \mathbb{F}_2$, where $1 \leq i \leq t$. The ADCA distinguisher is depicted as follows.

$$\delta_k^{\mathtt{ADCA}} = \mathtt{arg\ max} \#\{d_{alg}\left((f_k)_i\right) \leq d \mid i \in [t]\}$$

A degree-d ADCA detects the degrees of the Boolean functions $(f_k)_i$ and distinguishes the correct key with the maximum number of $d_{alg}\left((f_k)_i\right) \leq d$. When computing the degree of $(f_k)_i$, ADCA constructs a system of linear equations based on its ANF representations. Based on the d-th degree extension of the inputs x and each sample of traces, when the key guess is correct, the linear equations will be solvable by a degree-d ADCA such that $d_{alg}\left((f_k)_i\right) \leq d$.

3 Computation Analysis against SE-SPECK and IF-SPECK

Although SE-SPECK and IF-SPECK have been broken by algebraic attacks, they lack the security evaluations of computation analysis. The author of the implicit framework [RVP22] stated that the current automated gray-box attacks cannot be applied to break the implicit implementations, with the following statement:

> "These attacks usually target an intermediate computation where the output is encoded with a small function and depends on a few key bits. However, in implicit implementations, the only computations are evaluations of polynomials with large inputs and large encodings, and the outputs of these computations depend on the whole round keys. Finding new automated attacks to implicit implementations is an interesting challenge that we leave as future work."

In this section, we refine an encoded structure of SE-SPECK and IF-SPECK to discuss its resistance to computation analysis, such as DCA and ADCA.

3.1 Analysis of Encoded Structure

Both SE-SPECK and IF-SPECK apply the encoding to protect the inputs and outputs of round functions. SE-SPECK combines the self-equivalence encodings and the affine layer of SPECK to hide the subkeys. The resulting affine function is precomputed in the white-box implementation. Thus, the computation analysis cannot collect the intermediate values during the construction of the affine function. Similarly, IF-SPECK combines the round operation and the self-equivalence encoding as an implicit function. It computes the round outputs by solving an affine system. Hence, the attacker has no access to the intermediate values of the round functions. Based on these analyses, we suppose that the computation analysis can only analyze the traces containing the encoded round inputs/outputs.

By combining the first two encryption rounds, we can obtain a function F which has the same structure in both SE-SPECK and IF-SPECK. Let $x_0^{(1)}$ and $x_1^{(1)}$ denote an n-bit half of plaintexts, respectively. The n-bit first-round key is represented by $k^{(1)}$. The definition of F is described as follows. Figure 3 also depicts the structure of F.

$$\begin{aligned}
&F : (\mathbb{F}_2^n)^2 \mapsto (\mathbb{F}_2^n)^2 : F(x_0^{(1)}, x_1^{(1)}) = (y_0, y_1)\\
&y_0 = \left(\left((x_0^{(1)} \ggg \alpha) \boxplus x_1^{(1)}\right) \oplus k^{(1)}\right) \ggg \alpha\\
&y_1 = \left(\left((x_0^{(1)} \ggg \alpha) \boxplus x_1^{(1)}\right) \oplus k^{(1)}\right) \oplus (x_1^{(1)} \lll \beta)
\end{aligned}$$

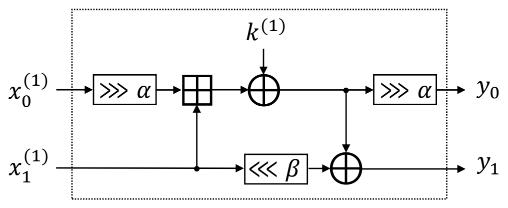

Figure 3: The same function F in the first two rounds of SE-SPECK and IF-SPECK.

In SE-SPECK, the third-round output can be computed as follows.

$$\begin{aligned}
(x_0^{(4)}, x_1^{(4)}) &= \overline{AL^{(2)}} \circ S \circ A^{(1)} \circ F(x_0^{(1)}, x_1^{(1)})\\
&= \overline{AL^{(2)}} \circ (B^{(1)})^{-1} \circ S \circ F(x_0^{(1)}, x_1^{(1)})\\
\overline{AL^{(2)}} &= A^{(2)} \circ AL^{(2)} \circ B^{(1)}
\end{aligned}$$

Let EC denote a combined function of the encoded affine layer $\overline{AL^{(2)}}$ and the self-equivalence $(B^{(1)})^{-1}$ such that $EC = \overline{AL^{(2)}} \circ (B^{(1)})^{-1}$. The third-round output of SE-SPECK can also be represented as

$$(x_0^{(4)}, x_1^{(4)}) = EC \circ S \circ F(x_0^{(1)}, x_1^{(1)}).$$

We note that EC can be generalized as a random affine encoding if the self-equivalence encoding $A^{(2)}$ is randomly generated in a large space.

The second-round output of IF-SPECK applies a modular addition S and an output encoding $EC = C^{(2)} \circ (B^{(1)})^{-1}$ to F. With the representation of an explicit function, its second-round outputs can be equivalently computed as

$$(x_0^{(3)}, x_1^{(3)}) = C^{(2)} \circ (B^{(1)})^{-1} \circ S \circ F(x_0^{(1)}, x_1^{(1)}) = EC \circ S \circ F(x_0^{(1)}, x_1^{(1)}).$$

The output encoding EC is affine if the self-equivalence $B^{(1)}$ is affine. However, when the quadratic self-equivalence is applied, the output encoding EC is non-linear because of the component of $(B^{(1)})^{-1}$. Particularly, the degree of EC is unknown. For achieving practical storage costs, the non-linear encoding EC needs to be constructed with low degrees, such as quadratic and cubic. Based on the above analysis, Figure 4 depicts the equivalent structure of the first three rounds in SE-SPECK and the first two rounds in IF-SPECK.

The equivalent structure can be refined as an encoded function F'_{k^*}, which is constructed by F followed by a modular addition function S and a random encoding EC. Let k^* denote the secret key of F'_{k^*}. By omitting the superscript of the first round, Figure 5

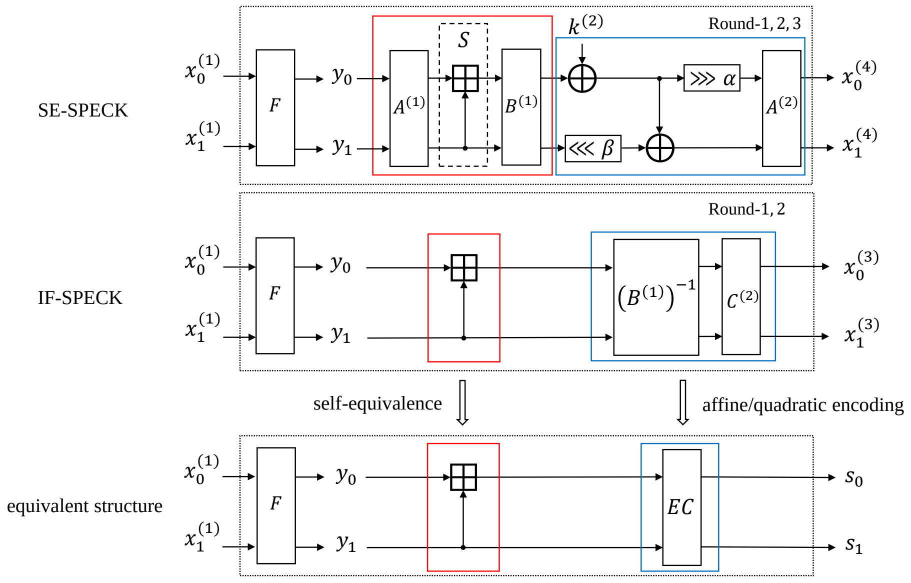

Figure 4: The equivalent structure of SE-SPECK and IF-SPECK.

illustrates the structure of the function F'_{k^*}, which can also be represented by the following equations.

$$
\begin{aligned}
F'_{k^*} &: (\mathbb{F}_2^n)^2 \mapsto (\mathbb{F}_2^n)^2 : F'_{k^*}(x_0, x_1) = EC \circ S \circ F(x_0, x_1) = EC(t_0, t_1) = (s_0, s_1) \\
t_0 &= y_0 \boxplus y_1 \\
&= ((((x_0 \ggg \alpha) \boxplus x_1) \oplus k^*) \ggg \alpha) \boxplus ((((x_0 \ggg \alpha) \boxplus x_1) \oplus k^*) \oplus (x_1 \lll \beta)) \\
t_1 &= y_1 = (((x_0 \ggg \alpha) \boxplus x_1) \oplus k^*) \oplus (x_1 \lll \beta)
\end{aligned}
$$

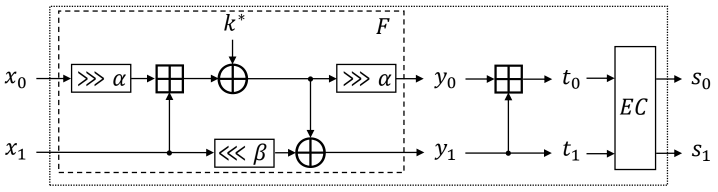

Figure 5: The structure of encoded function F'_{k^*} in SE-SPECK and IF-SPECK.

The vector t_0 can be deduced as a combination of a key addition and a substitution, namely the value of $S(x \oplus k^*)$. The vector t_1 can be regarded as a substitution followed by a key addition, i.e., the value of $S(x) \oplus k^*$. The output (s_0, s_1) applies a random encoding EC to the concatenation of the vectors t_0 and t_1. Based on this refined structure, we assume that the outputs (s_0, s_1) of SE-SPECK and IF-SPECK can be collected as the computation traces. For defeating IF-SPECK and SE-SPECK, the traces contain their second-round and third-round outputs, respectively. The computation analysis attempts to calculate the sensitive values (t_0, t_1) for different key guesses and to analyze the correlation between (t_0, t_1) and the encoded vectors (s_0, s_1).

3.2 DCA and ADCA against SE-SPECK and IF-SPECK

DCA. For the affine self-equivalences, the sensitive variables are obfuscated by the affine mappings. Bock *et al.* [BBMT18, BBB+19] analyzed the linear encoding against DCA. The results demonstrate that the rows with $\texttt{HW} = 1$ in the linear encoding will cause the key leakage. Different from CEJO with 8-bit small encodings, SE-SPECK and IF-SPECK apply at least 32-bit large encodings. Thus, the probability of constructing the affine encoding for SE-SPECK and IF-SPECK with $\texttt{HW} = 1$ is low. To counteract the linear encoding, Bock *et al.* suggested enumerating all the linear combinations of the sensitive values to fully recover the linear encodings. For such a brute-force DCA against SE-SPECK and IF-SPECK, DCA needs to enumerate the 2^{2n} linear combinations of the sensitive values. Let the key and the block sizes be n and $2n$, respectively. The time complexity of DCA against SE-SPECK and IF-SPECK with affine self-equivalence is $\mathcal{O}(t \cdot N \cdot 2^{3n})$. In some extreme cases, it requires $N = 2^{2n}$ plaintexts to generate 2^{2n} computation traces, which is too high to compute the correlations. For the quadratic self-equivalence cases, DCA is unlikely to obtain a high correlation due to the large non-linear encodings.

ADCA. For the target function $O_{k^*}(x) = EC \circ S \circ \oplus_{k^*}$ with an encoding EC in CEJO, ADCA constructs a set of inputs $x' = S^{-1}(x) \oplus k$ based on a key guess k. As ARX ciphers, SE-SPECK and IF-SPECK use the large modular addition for the substitution layer without the small Sbox. The output dimension of modular addition is n, which is different from the encoding size $2n$. Furthermore, the target function of SE-SPECK and IF-SPECK has a more complex structure. Therefore, it cannot trivially adapt the ADCA to attack SE-SPECK and IF-SPECK.

Discussion. Based on the above analysis, two significant challenges of computation analysis against SE-SPECK and IF-SPECK can be concluded as follows.

- Large inputs and key candidates. The non-linear layer of ARX ciphers consists of a modular addition. For attacking AES-like ciphers, DCA needs to compute a sensitive value, such as the Sbox output based on a key guess. Afterward, it computes the correlation between the trace and the computed sensitive value. Because of the large dimension of inputs and the large number of key candidates, the time complexities will be high in attacking SE-SPECK and IF-SPECK.

- Large encoding. Different from the 4/8-bit encodings in CEJO, white-box ARX ciphers are constructed by full-round encodings with at least 32-bit dimension. For an encoded ARX cipher with block size $2n$, it is impractical to compute the 2^{2n} linear combinations of the sensitive values to recover the affine encoding. Moreover, the implicit function can be constructed with quadratic encodings. It is infeasible for DCA to defeat the non-linear encodings. Although CEJO is vulnerable to DCA, it is still an open problem whether DCA can break the self-equivalence and implicit implementations with large affine or quadratic encoding.

4 Chosen-Plaintext Computation Analysis

The large encoding of self-equivalence and implicit structures hides the correlations between the sensitive values and the computation traces. To counteract encoding, it is straightforward to exhaustively search all possible constructed encodings of the selection function. If the key guess is correct, this process computes the encoded value which is highly correlated to the sample in the computation traces. However, it is impractical to perform this attack with a large block size. In this section, *chosen-plaintext computation* *analysis* (CP-CA) is introduced to reduce the affine encodings into a small linear space. Moreover, it can defeat the quadratic and other higher-degree non-linear encodings.

4.1 An Overview of CP-CA

The context of CP-CA follows the attack model of computation analysis. A CP-CA attacker constructs a reverse function with specific inputs to generate a set of target plaintexts. Subsequently, CP-CA invokes the white-box implementations with the chosen plaintexts and collects the computed intermediate values as computation traces. By applying the modeling analysis between the inputs and traces, CP-CA can extract the secret key of the white-box implementations.

Definition 9 (Reverse Function)**.** A reverse function is applied to choose the plaintexts of the cryptographic algorithm. It is constructed by some reverse steps of the cryptographic algorithm and needs to be instantiated by a key candidate.

Definition 10 (Attack Window)**.** Let $F_k(x)$ denote the n-bit target function with the input $x \in \mathbb{F}_2^n$, secret key $k \in \mathbb{F}_2^n$, and the n-bit output trace $y = F_k(x)$. An attack window $\mathcal{W}$ with size m ($m \leq n$) consists of a part of the target function such that $\mathcal{W}(x') = f_{k'}(x')$ for an m-bit mapping f with the partial input $x' \in \mathbb{F}_2^m$, partial key $k' \in \mathbb{F}_2^m$, and the m-bit output trace $y' = \mathcal{W}(x')$.

A CP-CA attack consists of the following three phases.

1. Partial key guess. CP-CA constructs a reverse function G with a partial key guess $k \in \mathcal{K}$. For instance, CP-CA can guess the partial key with nibble size. The function G maps the inputs $\mathcal{X}$ to a set of plaintexts $\mathcal{P}$. Subsequently, CP-CA invokes the cryptographic algorithm with the plaintext $p \in \mathcal{P}$ and collects the corresponding trace $\boldsymbol{v}$. By mounting the computation analysis on each attack window, CP-CA can distinguish a set of possible correct partial key candidates.

2. Partial key verification. For each two nibble key guesses, CP-CA extends the dimensions of the attack window to byte level to reduce the number of key candidates.

3. Round key verification. CP-CA generates a different reverse function G with various parameters to verify the correct key among the previously obtained key candidates.

In Phase 1, CP-CA computes the plaintext p from a reverse function G with chosen input x. We note that p consists of the output of G instead of its input. The motivations for the introduction of the reverse function are explained as follows.

- Reducing the input space. The reverse function will fix some bits of its inputs. Since CP-CA focuses on the modeling analysis between the chosen inputs (also the chosen intermediate values) and the computation traces, the input space of the CP-CA will be less than the block size $2n$.

- Reducing the encoding space. By fixing some input bits, the encoded output bits of the target function are correlated to a part of the input bits. Hence, the large encoding will be reduced to a small one.

4.2 The Reverse Function Construction

Different from CEJO, the encoded round outputs of the SE-SPECK and IF-SPECK depend on the whole round key. To reduce the key space, we introduce an instance of the reverse function G. It can calculate the unencoded sensitive values of the target function by choosing the inputs to G. Based on these unencoded values, CP-CA can focus on a small-dimension encoding to reduce the attack time complexity. Let $k \in \mathcal{K}$ denote a key guess for the reverse function and $k^* \in \mathbb{F}_2^n$ represent the embedded correct key for the cryptographic algorithm. Let (z_0, z_1) and (x_0, x_1) be the inputs and outputs of G, respectively, where the vectors (x_0, x_1) are also the plaintexts of the cryptographic algorithm. A key-initialized reverse function G_k is defined as follows.

$$
\begin{aligned}
&G_k : (\mathbb{F}_2^n)^2 \mapsto (\mathbb{F}_2^n)^2 : G_k(z_0, z_1) = (x_0, x_1)\\
&x_0 = ((((z_0 \boxminus z_1) \lll \alpha) \oplus k) \boxminus ((z_1 \oplus ((z_0 \boxminus z_1) \lll \alpha)) \ggg \beta)) \lll \alpha\\
&x_1 = (z_1 \oplus ((z_0 \boxminus z_1) \lll \alpha)) \ggg \beta
\end{aligned}
$$

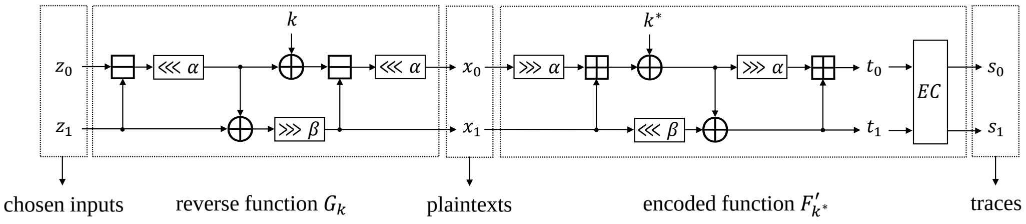

Figure 6: The reverse function of CP-CA against SE-SPECK and IF-SPECK.

Figure 6 also depicts such an application of a reverse function to the inputs of the encoded cryptographic algorithm. The computation of (x_0, x_1) contains the inverse operations from computing (t_0, t_1). To formalize the cancellation of the reverse function, the unencoded intermediate values (t_0, t_1) of the refined structure F'_{k^*} can be redefined as follows.

$$
\begin{aligned}
&F_{k^*} : (\mathbb{F}_2^n)^2 \mapsto (\mathbb{F}_2^n)^2 : F_{k^*}(x_0, x_1) = (t_0, t_1)\\
&t_1 = (((x_0 \ggg \alpha) \boxplus x_1) \oplus k^*) \oplus (x_1 \lll \beta)\\
&t_0 = ((((x_0 \ggg \alpha) \boxplus x_1) \oplus k^*) \ggg \alpha) \boxplus t_1
\end{aligned}
$$

The outputs (x_0, x_1) of the reverse function G_k can be computed as follows.

$$
\begin{aligned}
x_1 &= (z_1 \oplus ((z_0 \boxminus z_1) \lll \alpha)) \ggg \beta,\\
x_0 &= ((((z_0 \boxminus z_1) \lll \alpha) \oplus k) \boxminus x_1) \lll \alpha.
\end{aligned}
$$

With the composition of F_{k^*} and G_k, the sensitive values (t_0, t_1) can be derived from the inputs (z_0, z_1) as follows.

$$
\begin{aligned}
F_{k^*} \circ G_k : (\mathbb{F}_2^n)^2 \mapsto (\mathbb{F}_2^n)^2 &: F_{k^*} \circ G_k(z_0, z_1) = (t_0, t_1)\\
t_1 &= (((z_0 \boxminus z_1) \lll \alpha) \oplus k \oplus k^*) \oplus (z_1 \oplus ((z_0 \boxminus z_1) \lll \alpha))\\
&= z_1 \oplus k \oplus k^* \qquad (1)\\
t_0 &= ((((z_0 \boxminus z_1) \lll \alpha) \oplus k \oplus k^*) \ggg \alpha) \boxplus t_1\\
&= ((((z_0 \boxminus z_1) \lll \alpha) \oplus k \oplus k^*) \ggg \alpha) \boxplus (z_1 \oplus k \oplus k^*) \qquad (2)
\end{aligned}
$$

The reverse function is utilized to cancel out some first-round operations in the ARX-based white-box ciphers. The unencoded intermediate values in the following rounds can be precisely computed, which are the same values as the chosen inputs for the correct key guess. By elaborately selecting the intermediate values, CP-CA can mount the computation analysis on the collected traces.

4.3 Theoretical Analysis

Correct Key Guess. If the key guess k of the reverse function G_k is correct, we have $k = k^*$ such that $k \oplus k^* = 0$. The variables in Eq. (1) and Eq. (2) can be canceled out as

$$t_1 = z_1,\ t_0 = (((z_0 \boxminus z_1) \lll \alpha) \ggg \alpha) \boxplus z_1 = (z_0 \boxminus z_1) \boxplus z_1 = z_0.$$

For a correct key guess, the operations of the reverse function G_{k^*} and certain rounds of the cryptographic algorithm are canceled out. The unencoded intermediate values (t_0, t_1) are equal to the inputs (z_0, z_1) of G_{k^*}. CP-CA can distinguish the correct key by applying the computation analysis on the chosen inputs (z_0, z_1) (also the intermediate values (t_0, t_1)) and the computation traces. The collected traces are composed of the encoded values $(s_0, s_1) = EC(z_0, z_1)$ for an unknown affine or low-degree non-linear encoding EC. Figure 7 depicts the transformation from the inputs (z_0, z_1) of G_{k^*} to the outputs (s_0, s_1) of the encoded function F'_{k^*}.

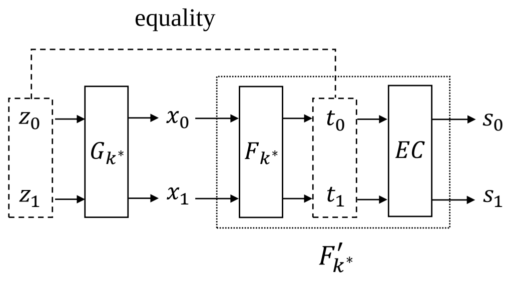

Figure 7: The transformation of a reverse function G_{k^*} for a correct key guess.

Incorrect Key Guess. For an incorrect key guess, the operations of the reverse function G_k and certain rounds of the cryptographic algorithm cannot be canceled out. The unencoded sensitive values (t_0, t_1) are computed by Eq. (1) and Eq. (2). It is a random function for different key candidates $k \neq k^*$. The computation of the encoded outputs (s_0, s_1) is an affine function or a low-degree non-linear mapping of (t_0, t_1) instead of (z_0, z_1). CP-CA cannot distinguish the correct key by analyzing the inputs (z_0, z_1) and the traces (s_0, s_1).

Affine Encoding. To deal with the affine encoding phase, CP-CA fixes half of the inputs such that $z_1 = c$. The n_b ($n_b < n$) bits of z_0 are the chosen inputs, while the most significant n_a bits and the least significant $n_c = n - n_a - n_b$ of z_0 are binary zero values. With the reverse function and the chosen inputs, the dimension of the affine encoding will be reduced, namely from 2^{2n} to 2^{n_b}. Therefore, CP-CA can mount the computation analysis for the dimension-reduced inputs and encodings.

Proposition 1. *Let $z \in \{0,1\}^{n_b}$ denote an n_b-bit vector, c be an n-bit constant. Let N_a and N_c represent an n_a-bit and an $n_c(= n - n_a - b_b)$-bit zero vector, respectively. Given an $2n$-bit affine function $EC : \mathbb{F}_2^{2n} \mapsto \mathbb{F}_2^{2n}$, the resulting vector $EC(N_a \parallel z \parallel N_c, c)$ can also be computed as $L' \cdot z \oplus l$, where L' is a $2n \times n_b$ matrix and $l \in \{0,1\}^{2n}$.*

Non-Linear Encoding. Based on the affine-quadratic self-equivalence, IF-SPECK can also be constructed with non-linear input/output encodings. Because of the chosen inputs, $n_a + n_c + n$ bits (N_a, N_c and c) of the inputs $b = (N_a \parallel z \parallel N_c, c)$ are constant values. The trace values y can be computed as $y = (s_0, s_1) = EC(b)$ for a low-degree non-linear encoding EC. Because of these fixed input bits, some variables in the monomials of the

ANF representations of EC are constants with degree 0. Accordingly, the degrees of the coordinate functions and the non-linear encoding EC will be reduced. For a $2n$-bit degree-d ($d \leq 2n-1$) non-linear encodings, each coordinate function y_i where $1 \leq i \leq 2n$ can be computed by ANF with the degree no more than d. We suppose that a degree-d monomial in its ANF is defined as $b_{i+1}b_{i+2}\cdot\cdots\cdot b_{i+d}$, where each b_{i+j} ($1 \leq j \leq d$) denotes a coordinate bit of the chosen input b. If any bit b_{i+j} is a constant of N_a, N_c or c such that $b_{i+j} = 0/1$, the monomial $b_{i+1}b_{i+2}\cdot\cdots\cdot b_{i+d}$ has a reduced degree which is less than d. Specifically, the ANF has the most degree n_b when the bits $\{b_{i+j}\}$ are all composed of the bits of z. For an instance of the quadratic encoding case, each coordinate function y_i for $1 \leq i \leq 2n$ can be computed by a degree-2 ANF. We suppose that a degree-2 monomial in its ANF is defined as $b_i \cdot b_j$ ($1 \leq i \neq j \leq 2n$), where b_i and b_j denote two different bits of the chosen input b. If one of b_i and b_j is a constant bit of N_a, N_c or c, the monomial $b_i \cdot b_j$ has a reduced degree which is less than 2. If b_i (resp. b_j) is a constant, it will be transformed as $(0/1) \cdot b_j$ (resp. $(0/1) \cdot b_i$). If both b_i and b_j are constant bits, the product $b_i \cdot b_j$ is a constant value. Only for the case that both b_i and b_j are the bits of z, the monomial $b_i \cdot b_j$ has degree 2.

4.4 Partial Key Recovery

Partial Key Guess. The reverse function in CP-CA is used to cancel out some operations in the encoded implementation. When the key guess is correct, the unencoded sensitive values are identical to the inputs of the reverse function. However, it is impractical to enumerate the key guess over the round key space. To reduce the key guess space, we introduce the phase of partial key guess into the CP-CA attack. The attacker can guess a nibble key ($2^4 = 16$ key candidates) and construct the corresponding reverse function to distinguish the possible correct key. When generating the reverse function, CP-CA fixes z_1 as 0, such that Eq. (1) and Eq. (2) can be transformed as

$$\begin{aligned} t_1 &= k \oplus k^* = \Delta k, \\ t_0 &= (((z_0 \lll \alpha) \oplus k \oplus k^*) \ggg \alpha) \boxplus (k \oplus k^*) \\ &= (z_0 \oplus (\Delta k \ggg \alpha)) \boxplus \Delta k. \end{aligned}$$

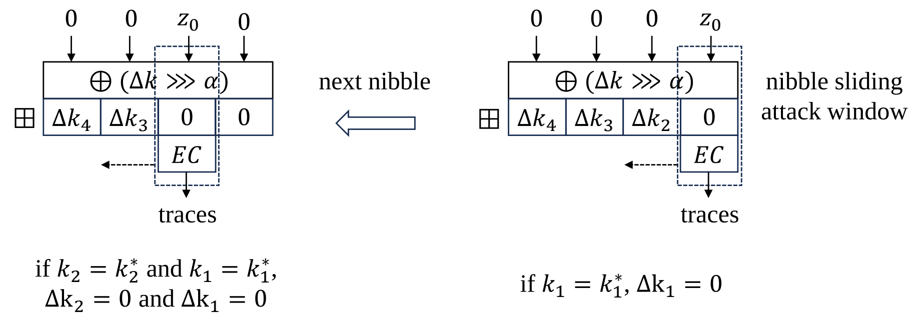
0
0
z_0
0
$\oplus (\Delta k \ggg \alpha)$
$\boxplus$
Δk_4
Δk_3
0
0
EC
traces
if $k_2 = k_2^*$ and $k_1 = k_1^*$,
$\Delta k_2 = 0$ and $\Delta k_1 = 0$
next nibble
0
0
0
z_0
$\oplus (\Delta k \ggg \alpha)$
$\boxplus$
Δk_4
Δk_3
Δk_2
0
EC
traces
nibble sliding
attack window
if $k_1 = k_1^*$, $\Delta k_1 = 0$

Figure 8: The illustration of the partial key guess in CP-CA.

During the encryption, the addition between the key guess k and the correct key k^* is a constant. If the key guess is correct, we have $\Delta k = 0$ such that $t_0 = z_0 \oplus (\Delta k \ggg \alpha)$. The sensitive value t_0 is linearly mapped from the chosen intermediate value z_0. On the contrary, t_0 will be transformed non-linearly with the chosen input z_0 for the carry of the modular addition. Based on these observations, CP-CA guesses the partial key from the least significant bits. Figure 8 depicts the attack process of the partial key guess. There is a sliding attack window with nibble size in both the key guess k and the chosen input z_0.

Specifically, CP-CA guesses the least significant nibble key at first. It then mounts the computation analysis between the chosen input nibble z_0 and the collected computation traces. We can obtain a set of possible key candidates for the least significant nibble bits. Subsequently, CP-CA combines the possible correct key guesses with the next nibble key guesses and applies the computation analysis on the next nibble chosen inputs. By iterating the above process, the round key will be fully recovered from the least significant nibble to the most significant one.

Partial Key Verification. According to the property of modular addition, some incorrect keys can also be falsely reported with the same first ranking as the correct ones. Moreover, each nibble key guess is concatenated with the previously recovered key candidates. The space might be large for the most significant nibble key. Since more traces might enhance the results in the power analysis, the attacker can extend the dimension of the attack window, i.e., to mount a CP-CA with larger n_b and N. Figure 9 illustrates the process of partial key verification. For each two recovered nibble key candidates, CP-CA applies the computation analysis between the chosen input byte z_0 and the traces to verify the correctness of the byte key guess. Consequently, the verified keys will be combined with the next nibble key guess to recover the round key. Figure 10 depicts the alternated process of the nibble key guess and byte key verification.

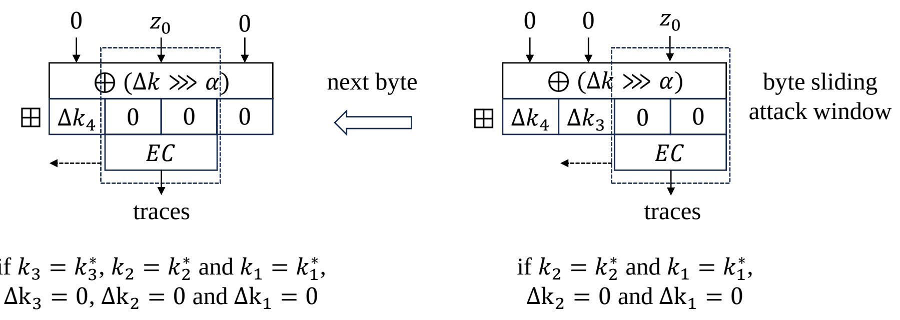
0
z_0
0
$\oplus (\Delta k \ggg \alpha)$
$\boxplus$
Δk_4
0
0
0
EC
traces
next byte
0
0
z_0
$\oplus (\Delta k \ggg \alpha)$
$\boxplus$
Δk_4
Δk_3
0
0
EC
traces
byte sliding attack window
if $k_3 = k_3^*$, $k_2 = k_2^*$ and $k_1 = k_1^*$, $\Delta k_3 = 0$, $\Delta k_2 = 0$ and $\Delta k_1 = 0$
if $k_2 = k_2^*$ and $k_1 = k_1^*$, $\Delta k_2 = 0$ and $\Delta k_1 = 0$

Figure 9: Process of partial key verification.

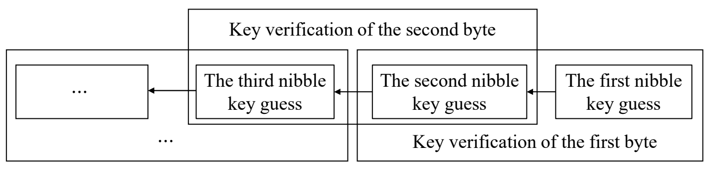
Key verification of the second byte
...
The third nibble key guess
The second nibble key guess
The first nibble key guess
...
Key verification of the first byte

Figure 10: Alternated process of the partial key guess and verification.

Round Key Verification. After the last byte key verification, we can obtain a small set of round key candidates. To distinguish the correct key, we introduce a round key verification phase into CP-CA. It constructs the reverse function with different parameters, especially the chosen constant inputs. For an incorrect key guess, the different constants c in the chosen inputs $(z_0, z_1) = (N_a \parallel z \parallel N_c, c)$ generate different intermediate values (t_0, t_1). If $c = 0$, the non-linear modular subtractions in the computation of t_0 (refer to Eq. (2)) will be canceled out. Moreover, $\Delta k = k \oplus k^*$ has low `HW` when the incorrect key k is similar to the secret key k^*. The vector t_0 is similar (or equal) to the input z_0. Thus, it fails to distinguish the correct key. Based on this observation, we suggest instantiating a non-zero constant for z_1 such that CP-CA distinguishes the correct round key with a nibble chosen input z and a non-zero constant input c.

4.5 Master Key Recovery

After extracting the first-round key, CP-CA needs to construct another reverse function corresponding to the next round. Moreover, the reverse function is initialized with the recovered keys. Suppose that the size of the master key is $m \cdot n$, where n is the half of the block size and m is the number of key words. If $k^{(1)}, \cdots k^{(m)}$ are known, we can compute

$$\begin{aligned} l^{(r+m-2)} &= (k^{(r)} \lll \beta) \oplus k^{(r+1)}, \\ l^{(r)} &= \Big(\Big(l^{(r+m-2)} \oplus (r-1)\Big) \boxminus k^{(r)}\Big) \lll \alpha. \end{aligned}$$

The master key is composed of $l^{(m-1)}, \cdots, l^{(1)}$, and $k^{(1)}$. We note that each round key can be recovered separately. The time complexity of the master key recovery is m times of the round key recovery. Since the key schedule of SPECK is invertible, a full key recovery can be deduced from the master key.

5 The CP-CA Distinguishers

For a concrete analysis, this section first proposes an attack instance of CP-DCA, which reconstructs the DCA attack following the CP-CA model. Subsequently, a new *chosen-plaintext linear encoding analysis* (CP-LEA) is proposed as a reformulation of ADCA to recover the linear system between the chosen intermediate values and a part of the traces. Moreover, CP-DCA and CP-LEA are extended as higher-degree attacks to break the non-linear encodings.

5.1 Chosen-Plaintext DCA

Affine Encoding. As a computation analysis, CP-DCA calculates the correlation between the chosen intermediate values and the traces. The key guess with the maximum number of the highest correlation is the most likely correct.

Corollary 1. *Let y_i $(1 \le i \le 2n)$ denote the output coordinate of $EC(N_a \parallel z \parallel N_c, c)$. There exist $2n$ linear combinations L of z satisfying* $|\texttt{Cor}(L \cdot z, y_i)| = 1$.

Proof. Proposition 1 illustrates that the output y_i can be represented as a product of the row in L' and the input z, followed by a constant addition. It indicates that y_i can be computed by an affine function of z. If the constant l is zero, the computation of y_i is a linear function of z. Thus, we have $|\texttt{Cor}(L \cdot z, y_i)| = 1$ where L is one of the $2n$ row vectors of L'. Since a constant addition with a binary value is equal to a bit flip of the vector, the computed correlation will not change. This implies that $|\texttt{Cor}(L \cdot z, y_i)| = 1$ is still satisfied in a non-zero constant case. The corresponding $2n$ linear combinations L are equivalent to the $2n$ row vectors of L'. □

Based on Corollary 1, the outputs (s_0, s_1) of the encoded function F'_{k^*} (refer to Figure 7) can be computed by a linear combination of the chosen input z. Ideally, $2n$ correlations 1 can be calculated between $2n$ linear combinations L of z and the traced values (s_0, s_1). Hence, CP-DCA enumerates all 2^{n_b} linear combinations of the chosen input z and looks for a key guess with the maximum number (most likely $2n$) of the highest correlation (most likely 1) as the correct key candidate. Let (z_0, z_1) denote two n-bit inputs of the reverse function G_k for a key guess k. To reduce the vector space of the chosen inputs, CP-DCA fixes half of the inputs such that $z_1 = c$ $(c \neq 0)$ for an n-bit constant c. The input

$z_0 = (N_a \parallel z \parallel N_c) = (N_a \parallel \{0,1\}^{n_b} \parallel N_c)$ for $n_b = n - n_a - n_c$, where N_a and N_c denote an n_a-bit and n_c-bit binary zero value, respectively. Thus, the CP-DCA attacker only chooses the n_b bits out of the n bits in z_0. The $2n$-bit plaintexts (x_0, x_1) are computed as

$$(x_0, x_1) = G_k(z_0, z_1) = G_k\left(N_a \parallel z \parallel N_c, c\right).$$

Let $z^{(j)} \in \{0,1\}^{n_b}$ $(1 \le j \le N)$ denote the j-th n_b-bit chosen input of G_k among N inputs. For each key guess k, CP-DCA collects N t-sample computation traces $\boldsymbol{v}^{(j)} = \{v_1^{(j)}, v_2^{(j)}, \cdots, v_t^{(j)}\}$ which contain the outputs of the function $G_k(N_a \parallel z^{(j)} \parallel N_c, c)$. For the principal of the reverse function, the sensitive value is equal to the chosen input $z^{(j)}$ if the key guess is correct. Thus, CP-DCA enumerates the 2^{n_b} linear combinations L of $z^{(j)}$ and calculates the correlation between the resulting binary value $L \cdot z^{(j)}$ and each sample $v_i^{(j)}$ for $1 \le i \le t$ of the traces $\boldsymbol{v}^{(j)}$. Let k^* denote the correct key and $\overline{E_{k^*}}$ be the white-box cryptographic algorithm. Figure 11 illustrates the construction of G_k and the workflow of a CP-DCA distinguisher.

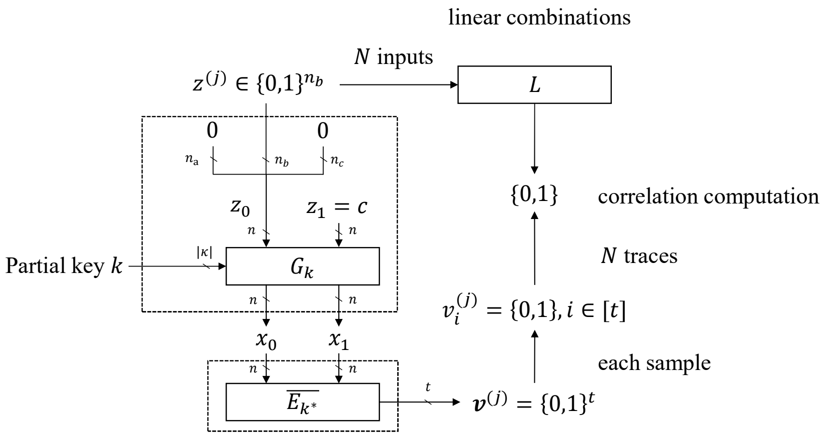

Figure 11: Reverse function G_k and CP-DCA distinguisher.

The computation of linear combinations of the bits in $z^{(j)}$ aims to calculate all possible linear encodings of the sensitive values (t_0, t_1) (refer to Figure 7). For a $2n$-bit affine encoding, the values (t_0, t_1) are encoded by $2n$ linear combinations corresponding to the $2n$ rows of the matrix in the affine encoding. Hence, $2n$ samples of the traces can be linearly represented by the bits of the sensitive value for the correct key case. If the key guess is incorrect, less than $2n$ samples of the traces are correlated to the linear combinations of the sensitive values. Based on these analyses, the CP-DCA distinguisher selects the key guess with the maximum number of the highest correlation as the most likely correct one. The CP-DCA distinguisher $\delta_k^{\mathtt{CP-DCA}}$ is defined as follows.

$$\delta_k^{\mathtt{CP-DCA}} = \mathtt{arg\ max}\ \#\left\{\mathtt{max}\left|\mathtt{Cor}(L \cdot z^{(j)}, v_i^{(j)})\right|\right\}$$

Time Complexity. Algorithm 1 describes the detailed procedures of CP-DCA. It requires a reverse function G and N n_b-bit chosen inputs $z^{(j)}$ for $1 \le j \le N$. The algorithm contains the three processes as illustrated as follows.

- Traces collection. For each key guess k, CP-DCA encrypts N chosen plaintexts (computed by N chosen inputs $z^{(j)}$) and collects the corresponding t-sample computation traces $\boldsymbol{v}^{(j)} = (v_1^{(j)}, v_2^{(j)}, \cdots, v_t^{(j)})$. The time complexity of this process is $\mathcal{O}(|\mathcal{K}| \cdot N)$, where $\mathcal{K}$ is the key space.

Algorithm 1 The procedure of CP-DCA

```
Input: $n_a, n_c$ ($n_a, n_c \leq n$): the vector space of the zero input values
       $n_b = n - n_a - n_c$: the vector space of the chosen inputs
       $c$: an $n$-bit constant vector
       $N$ inputs $z^{(j)} \in \{0, 1, \cdots, 2^{n_b} - 1\}$ for $1 \leq j \leq N$
       a reverse function $G : (\mathbb{F}_2^n)^2 \mapsto (\mathbb{F}_2^n)^2$
Output: a set of the probably correct keys $\mathcal{K}'$
 1: $\mathcal{K}' \leftarrow \emptyset$
 2: $k_{\texttt{count}} \leftarrow 0$
 3: $k_{\texttt{max}} \leftarrow 0$
 4: for each $k \in \mathcal{K}$ do                                        ▷ each key guess
 5:     for $i \leftarrow 1, N$ do                                        ▷ chosen plaintexts
 6:         $\boldsymbol{v}^{(j)} = (v_1, v_2, \cdots, v_t) \leftarrow \overline{E_{k^*}}(G_k(N_a \parallel z^{(j)} \parallel N_c, c))$   ▷ traces collection
 7:     $l_{\texttt{count}} \leftarrow 0$
 8:     $l_{\texttt{max}} \leftarrow 0$
 9:     $L_{\texttt{score}} \leftarrow$ empty associative array
10:     for each $L \in \{1, 2, \cdots, 2^{n_b} - 1\}$ do       ▷ enumerations of linear combination
11:         $s_{\texttt{max}} \leftarrow 0$
12:         for $i \leftarrow 1, t$ do                                 ▷ each sample of traces
13:             for $j \leftarrow 1, N$ do
14:                 $\{v_i\} \leftarrow (\boldsymbol{v}^{(j)})_i$
15:                 $\{y_s\} \leftarrow L \cdot z^{(j)}$
16:             score $\leftarrow \texttt{Cor}(\{y_s\}, \{v_i\})$               ▷ correlation computation
17:             if score $> s_{\texttt{max}}$ then
18:                 $s_{\texttt{max}} \leftarrow$ score
19:         $L_{\texttt{score}}[L] \leftarrow s_{\texttt{max}}$
20:         if $L_{\texttt{score}}[L] > l_{\texttt{max}}$ then
21:             $l_{\texttt{max}} \leftarrow L_{\texttt{score}}[L]$
22:     $k_{\texttt{score}} \leftarrow l_{\texttt{max}}$
23:     if $k_{\texttt{score}} = k_{\texttt{max}}$ then                 ▷ searching for the highest correlation
24:         for each $L \in \{1, 2, \cdots, 2^{n_b} - 1\}$ do
25:             if $L_{\texttt{score}}[L] = l_{\texttt{max}}$ then
26:                 $l_{\texttt{count}} \leftarrow l_{\texttt{count}} + 1$
27:         if $l_{\texttt{count}} > k_{\texttt{count}}$ then
28:             $k_{\texttt{count}} \leftarrow l_{\texttt{count}}$
29:             $\mathcal{K}' \leftarrow \{k\}$
30:         if $l_{\texttt{count}} = k_{\texttt{count}}$ then
31:             $\mathcal{K}' \leftarrow \mathcal{K}' \cup \{k\}$
32:     if $k_{\texttt{score}} > k_{\texttt{max}}$ then
33:         $k_{\texttt{max}} \leftarrow k_{\texttt{score}}$
34:         for each $L \in \{1, 2, \cdots, 2^{n_b} - 1\}$ do
35:             if $L_{\texttt{score}}[L] = l_{\texttt{max}}$ then
36:                 $l_{\texttt{count}} \leftarrow l_{\texttt{count}} + 1$
37:         $k_{\texttt{count}} \leftarrow l_{\texttt{count}}$          ▷ the maximum number of the highest correlation
38:         $\mathcal{K}' \leftarrow \{k\}$
39: return $\mathcal{K}'$
```

- Correlation computation. CP-DCA computes 2^{n_b} linear combinations L of the bits in the chosen inputs $z^{(j)}$ and calculates the correlation between the resulting binary value $L \cdot z^{(j)}$ and each sample of traces $v_i^{(j)}$ for $1 \leq i \leq t$ and $1 \leq j \leq N$. This process is related to the size of key candidates $|\mathcal{K}|$, the number of linear combinations 2^{n_b}, the number of samples t, and the number of plaintexts/traces N. Thus, this process has the time complexity $\mathcal{O}(|\mathcal{K}| \cdot 2^{n_b} \cdot t \cdot N)$.

- Searching for the highest correlation. For each key guess, CP-DCA computes the correlations for 2^{n_b} linear combinations. Subsequently, CP-DCA searches for the key with the maximum number of the highest correlation as the most likely correct key. This process has the time complexity $\mathcal{O}(|\mathcal{K}| \cdot 2^{n_b})$.

Based on the above analyses, the time complexity of CP-DCA depends on the process of correlation computation. Due to the partial key guess phase, CP-DCA needs to iterate n/n_b times to recover the round key. Thus, the total time complexity is $\mathcal{O}(n/n_b \cdot |\mathcal{K}| \cdot 2^{n_b} \cdot t \cdot N)$.

Non-Linear Encoding. To break the non-linear masking, *higher-degree higher-order* DCA (HDHO-DCA) was proposed by Tang *et al.* [TGCX23] at CHES 2023. A degree-d and order-n HDHO-DCA multiplies all d points in the computation trace and then combines every n nodes in the higher-degree trace. It aims to fully recover both the non-linear and linear encodings. Following the CP-CA model, we reformulate HDHO-DCA as *higher-degree computation analysis* (HDCA) to deal with the non-linear encoding of ARX-based white-box ciphers. Different from HDHO-DCA, CP-HDCA extends the higher-degree bits of the chosen inputs instead of the computation traces. A degree-d CP-HDCA computes all the degree-l ($2 \leq l \leq d$) extensions of the bits of the chosen inputs $z_d^{(j)}$. By combining the original n_b input bits, the extension consists of $q = \binom{n_b}{1} + \cdots + \binom{n_b}{d}$ items. CP-HDCA computes 2^q linear combinations L of the extended bits $z_d^{(j)}$ and calculates the correlation between the resulting binary value $L \cdot z_d^{(j)}$ and each sample of traces $v_i^{(j)}$ for $1 \leq i \leq t$ and $1 \leq j \leq N$. Thus, the time complexity of the correlation computation will be $\mathcal{O}(|\mathcal{K}| \cdot 2^q \cdot t \cdot N)$ while the one of searching for the highest correlation will be $\mathcal{O}(|\mathcal{K}| \cdot 2^q)$.

5.2 Chosen-Plaintext Linear Encoding Analysis

Both CP-DCA and CP-HDCA enumerate the encodings with the exponential time complexity. Particularly, CP-HDCA is impractical for the quadratic encoding with the time complexity $\mathcal{O}(2^{52} \cdot t)$ when $n_b = 8$, $d = 2$, $|\mathcal{K}| = 2^8$, and $N = 2^8$ in the byte-level partial key verification phase. To reduce the time complexity, we reformulate ADCA as CP-LEA following the CP-CA model. CP-LEA recovers the linear systems between the chosen intermediate values and a part of the computation traces to distinguish the correct key.

Affine Encoding. Due to the reverse function reducing the large linear encoding into a small space, we combine the technique of ADCA and the model of CP-CA as CP-LEA to recover the linear mappings from the chosen intermediate values to a part of the traces. CP-LEA constructs a linear system that consists of the coordinates of z and each sample of $\boldsymbol{v}$. The key guess with the maximum number of solvable linear systems is the most likely correct one.

Corollary 2. *Let y_i ($1 \leq i \leq 2n$) denote the output coordinate of $EC(N_a \parallel z \parallel N_c, c)$. Let Z_j ($1 \leq j \leq n_b$) denote the bits of z. There exist $2n$ vectors $a = (a_0, a_1, \cdots, a_{n_b})$ satisfying*

$$[1 \ Z_1 \ \cdots \ Z_{n_b}] \cdot a^T = y_i, \text{ for } 1 \leq i \leq 2n.$$

Proof. Proposition 1 reveals that the output y_i can be represented as a product of a row in L' and the input z, followed by a constant addition. It indicates that y_i can be computed by an affine function of z. Let $(a_1, \cdots, a_{n_b})$ denote a row vector of L'. Let a_0 be a constant bit. Each output y_i can be represented as

$$y_i = (a_1, \cdots, a_{n_b}) \cdot z \oplus a_0.$$

If the constant a_0 is zero, the computation of y_i will be a linear function of z. The $2n$ vectors $(a_1, \cdots, a_{n_b})$ are equivalent to the $2n$ row vectors of L'. The corresponding $2n$ constant bits a_0 form the constant vector l of the affine mapping. The vector $(a_0, a_1, \cdots, a_{n_b})$ also corresponds to the coefficients of the ANF of the affine mapping. □

Based on Corollary 2, the outputs (s_0, s_1) of the encoded function F'_{k^*} (refer to Figure 7) can be computed by a linear system of the chosen input z. Moreover, $2n$ linear systems can be solved because of the linear mappings from the chosen inputs z to the trace values (s_0, s_1). Hence, CP-LEA looks for a key guess with the maximum number (most likely $2n$) of solvable linear systems as the correct key candidate. Let $z^{(j)} \in \{0,1\}^{n_b}$ $(1 \leq j \leq N)$ denote the j-th n_b-bit chosen input of G_k among N inputs. For each partial key guess k, CP-LEA collects N t-sample computation traces $\boldsymbol{v}^{(j)} = \{v_1^{(j)}, v_2^{(j)}, \cdots, v_t^{(j)}\}$ which contain the outputs of the function $G_k(N_a \parallel z^{(j)} \parallel N_c, c)$. For a correct key guess, the trace values (s_0, s_1) are computed as the encoded values $EC(N_a \parallel z^{(j)} \parallel N_c, c)$ for an affine encoding encoding EC. Since the trace values $v_i^{(j)}$ $(1 \leq i \leq t)$ are linearly mapped from the chosen inputs $z^{(j)}$, there exists a constant vector $a = (a_0, a_1, \cdots, a_{n_b})$ satisfying

$$[1 \ Z_1^{(j)} \ \cdots \ Z_{n_b}^{(j)}] \cdot a^T = v_i^{(j)},$$

where $Z_i^{(j)}$ for $1 \leq i \leq n_b$ denotes the coordinate of $z^{(j)}$, and $v_i^{(j)}$ for $1 \leq i \leq t$ is a trace sample. For N chosen inputs and each sample of traces, the attacker can construct a linear system. If the system is solvable, the trace values can be represented as a linear combination of the chosen inputs. It implies that the key guess is correct and the resulting vector $(a_0, a_1, \cdots, a_{n_b})$ recovers the equivalent linear encoding. Figure 12 illustrates the construction of G_k and the workflow of the CP-LEA distinguisher.

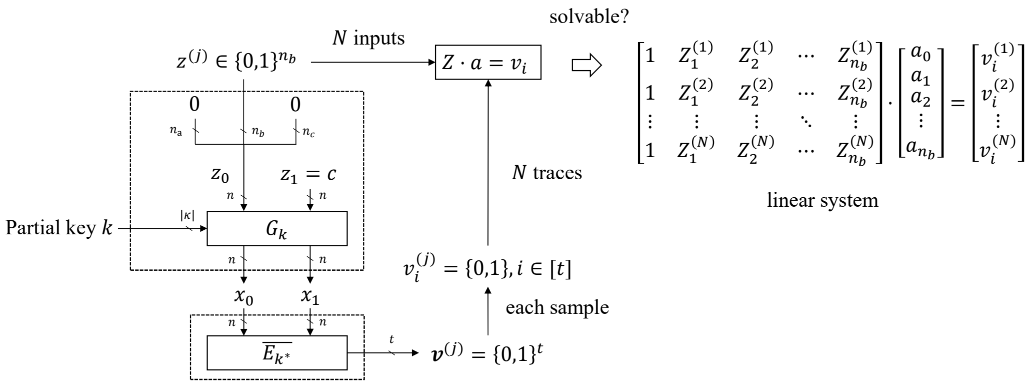
solvable?
N inputs
$z^{(j)} \in \{0,1\}^{n_b}$
$Z \cdot a = v_i$
$\begin{bmatrix} 1 & Z_1^{(1)} & Z_2^{(1)} & \cdots & Z_{n_b}^{(1)} \\ 1 & Z_1^{(2)} & Z_2^{(2)} & \cdots & Z_{n_b}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & Z_1^{(N)} & Z_2^{(N)} & \cdots & Z_{n_b}^{(N)} \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n_b} \end{bmatrix} = \begin{bmatrix} v_i^{(1)} \\ v_i^{(2)} \\ \vdots \\ v_i^{(N)} \end{bmatrix}$
linear system
0
0
n_a
n_b
n_c
z_0
$z_1 = c$
N traces
Partial key k
$|\kappa|$
G_k
$v_i^{(j)} = \{0,1\}, i \in [t]$
each sample
x_0
x_1
$\overline{E}_{k^*}$
t
$\boldsymbol{v}^{(j)} = \{0,1\}^t$

Figure 12: Reverse function G_k and CP-LEA distinguisher.

The constructed linear system aims to calculate an equivalent linear encoding of the affine encoding EC. Since the trace values are linearly mapped from the chosen intermediate values, the linear system is solvable for the correct key guess. If the key guess is incorrect, the linear system has no solution because the trace value is randomly mapped from the chosen inputs. The trace value cannot be represented by a linear mapping of the chosen inputs. Therefore, the CP-LEA distinguisher selects the key guess with the maximum number of solvable linear systems as the most likely correct key. Let Z denote the coefficients of the linear system which consists of the coordinates of the chosen inputs. Let v_i denote the vector of the same sample of different traces. The constructed linear system can be represented as $Z \cdot a = v_i$ for a possible solution vector a. Let $r(Z)$ denote the rank of the matrix Z. The linear system is solvable if $r(Z) \geq r(Z \mid v_i)$. The CP-LEA distinguisher $\delta_k^{\texttt{CP-LEA}}$ is defined as follows.

$$\delta_k^{\texttt{CP-LEA}} = \texttt{arg max} \ \# \{r(Z) \geq r(Z \mid v_i)\}$$

We note that only the trace vector v_i is related to the key guess. Since the computation of the coefficients Z of the linear system is irrelevant to the key candidates and the traces, CP-LEA can precompute Z and its rank. The row reductions of Z by Gauss Elimination can also be stored beforehand. In the attack process, the prestored operational steps can be applied to the trace vector v_i to calculate $r(Z \mid v_i)$. Algorithm 2 describes the detailed procedures of CP-LEA. It requires a reverse function G and N n_b-bit chosen inputs $z^{(j)}$ for $1 \leq j \leq N$. The algorithm contains the three processes as follows.

Algorithm 2 The procedure of CP-LEA

```
Input: $n_a, n_c$ ($n_a, n_c \leq n$): the vector space of the zero input values
       $n_b = n - n_a - n_c$: the vector space of the chosen inputs ($n_b \leq n$)
       $c$: an $n$-bit constant vector
       $N$ inputs $z^{(j)} \in \{0, 1, \cdots, 2^{n_b} - 1\}$ for $1 \leq j \leq N$
       $r(Z)$: the rank of the coefficient matrix $Z$
       $R(Z)$: the steps of the row reductions of $Z$
       a reverse function $G : (\mathbb{F}_2^n)^2 \mapsto (\mathbb{F}_2^n)^2$
Output: a set of the probably correct keys $\mathcal{K}'$
 1: $\mathcal{K}' \leftarrow \emptyset$
 2: $k_{\texttt{max}} \leftarrow 0$
 3: for each $k \in \mathcal{K}$ do                                    ▷ each key guess
 4:     $k_{\texttt{count}} \leftarrow 0$
 5:     for $j \leftarrow 1, N$ do                                     ▷ chosen plaintexts
 6:         $\boldsymbol{v}^{(j)} = (v_1, v_2, \cdots, v_t) \leftarrow \overline{E_{k^*}}(G_k(N_a \parallel z^{(j)} \parallel N_c, c))$   ▷ computation traces
 7:     for $i \leftarrow 1, t$ do                                     ▷ each sample of traces
 8:         for $j \leftarrow 1, N$ do
 9:             $\{v_i\} \leftarrow (\boldsymbol{v}^{(j)})_i$
10:         $v_i \leftarrow R(Z) \Rightarrow r(Z \mid v_i)$                ▷ row reduction on $v_i$
11:     if $r(Z) \geq r(Z \mid v_i)$ then                              ▷ solvable linear system
12:         $k_{\texttt{count}} \leftarrow k_{\texttt{count}} + 1$
13: if $k_{\texttt{count}} > k_{\texttt{max}}$ then               ▷ the maximum number of solvable linear systems
14:     $k_{\texttt{max}} \leftarrow k_{\texttt{count}}$
15:     $\mathcal{K}' \leftarrow \{k\}$
16: if $k_{\texttt{count}} = k_{\texttt{max}}$ then
17:     $\mathcal{K}' \leftarrow \mathcal{K}' \cup \{k\}$
18: return $\mathcal{K}'$
```

- Traces collection. For each key guess k, CP-LEA encrypts N chosen plaintexts (computed by N chosen inputs $z^{(j)}$) and collects the corresponding t-sample computation traces $\boldsymbol{v}^{(j)} = (v_1^{(j)}, v_2^{(j)}, \cdots, v_t^{(j)})$. The time complexity of this process is $\mathcal{O}(|\mathcal{K}| \cdot N)$, where $\mathcal{K}$ is the key space and $|\mathcal{K}|$ denotes the number of key candidates.

- Computation of linear systems. CP-LEA computes a linear system of the coordinates of the chosen inputs and the trace vectors. For different key guesses and each sample of traces, CP-LEA needs to verify the solvability of the linear equations $Z \cdot a = v_i$. Since the row reductions of the coefficients are precomputed, the maximum number of the operational steps for calculating $r(Z \mid v_i)$ are $N \cdot (n_b + 1)$. Thus, this process has the time complexity $\mathcal{O}(|\mathcal{K}| \cdot t \cdot N \cdot (n_b + 1))$.
- Searching for the solvable linear systems. For each key guess, CP-LEA computes the number of solvable linear systems. It selects the key guess with the computed maximum number as the correct candidate. This process has the time complexity $\mathcal{O}(|\mathcal{K}|)$.

Based on the above analysis, the time complexity of CP-LEA depends on the process of the computation of linear systems. Due to the partial key guess phase, CP-LEA needs to iterate n/n_b times to recover the round key. Thus, the total time complexity is $\mathcal{O}(n/n_b \cdot |\mathcal{K}| \cdot t \cdot N \cdot (n_b + 1))$.

Non-Linear Encoding. To deal with degree-d ($d \leq n_b$) non-linear encoding, we extend LEA as a higher-degree attack which is called *higher-degree encoding analysis* (HDEA). Similar to CP-HDCA, a degree-d CP-HDEA computes all the degree-l ($2 \leq l \leq d$) extensions of the bits of the chosen input $z^{(j)}$. With the original input $(Z_1^{(j)}, Z_2^{(j)}, \cdots, Z_{n_b}^{(j)})$, we can obtain the degree-d extended bits

$$\mathcal{Z}_d^{(j)} = \{1, Z_1^{(j)}, Z_2^{(j)}, \cdots, Z_{n_b}^{(j)}, Z_1^{(j)} Z_2^{(j)}, \cdots, Z_{n_b-1}^{(j)} Z_{n_b}^{(j)}, \cdots, Z_{n_b-d+1}^{(j)} \cdots Z_{n_b-1}^{(j)} Z_{n_b}^{(j)}\}.$$

The cardinality of the set $\mathcal{Z}_d^{(j)}$ is $p = \binom{n_b}{0} + \binom{n_b}{1} + \cdots + \binom{n_b}{d}$. Since EC is a degree-d function of the chosen inputs $z^{(j)}$, the trace value $y = EC(b)$ can be represented by some (unknown) linear combinations of $\mathcal{Z}_d^{(j)}$. Thus, the ANF of y_i can be represented as follows.

$$y_i(z^{(j)}) = a_0 \oplus a_1 Z_1^{(j)} \oplus a_2 Z_2^{(j)} \oplus \cdots \oplus a_{p-1} Z_{n_b-d+1}^{(j)} \cdots Z_{n_b-1}^{(j)} Z_{n_b}^{(j)},$$

where $a_0, a_1, \cdots, a_{p-1}$ are the coefficients over $\mathbb{F}_2$. Consequently, a degree-d CP-HDEA can solve the following linear equations.

$$\begin{pmatrix} 1 & Z_1^{(1)} & Z_2^{(1)} & \cdots & Z_{n_b-d+1}^{(1)} \cdots Z_{n_b-1}^{(1)} Z_{n_b}^{(1)} \\ 1 & Z_1^{(2)} & Z_2^{(2)} & \cdots & Z_{n_b-d+1}^{(2)} \cdots Z_{n_b-1}^{(2)} Z_{n_b}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & Z_1^{(N)} & Z_2^{(N)} & \cdots & Z_{n_b-d+1}^{(N)} \cdots Z_{n_b-1}^{(N)} Z_{n_b}^{(N)} \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{p-1} \end{pmatrix} = \begin{pmatrix} v_i^{(1)} \\ v_i^{(2)} \\ \vdots \\ v_i^{(N)} \end{pmatrix}$$

Degree-d CP-HDEA computes a linear system of the degree-d extension of the input coordinates and the trace vectors. For different key guesses and each sample of traces, CP-HDEA needs to verify the solvability of the linear equations $Z \cdot a = v_i$. Since the row reductions of the coefficients are precomputed, the maximum number of the operational steps for calculating $r(Z \mid v_i)$ are $N \cdot p$. Thus, the time complexity for the computation of linear systems will be $\mathcal{O}(|\mathcal{K}| \cdot t \cdot N \cdot p)$.

6 Experimental Results and Comparisons

6.1 Simulations

CP-CA can be instantiated with different parameters. For block size $2n$, CP-CA constructs N chosen inputs as $(N_a \parallel z \parallel N_c, c)$ where N_a and N_c represent n_a-bit ($n_a < n$) and

n_c-bit ($n_c < n$) zero values, respectively. The input c is an n-bit constant. The chosen input z is an n_b-bit ($n_b = n - n_a - n_c$) value. Table 2 illustrates the feasible parameters of CP-DCA, CP-LEA, and degree-2 CP-HDEA. The CP-DCA/LEA and CP-HDEA focus on counteracting the affine and quadratic encodings, respectively. In the partial key guess and verification phases, the input c is fixed as a zero value such that $c = \mathtt{0}$. Contrarily, the input c is fixed by a non-zero value $\mathtt{1}$ in the round key verification phase. Since both the partial key guess and the round key verification phases have a nibble sliding window with $n_b = 4$, the two phases choose the input z and guess the key with nibble size. The $n_b = 8$ in the partial key verification phase implies that the chosen input has a byte size. CP-DCA computes the correlation between the chosen intermediate values and traces while CP-LEA/HDEA constructs a (higher-degree extended) linear system to recover the mapping from the chosen intermediate values to a part of the traces. To increase the capability with more traces, CP-DCA enumerates the chosen inputs such that $N = 2^{n_b}$. Based on the Gauss Elimination, we suggest using the chosen inputs z for CP-LEA and CP-HDEA in Table 2, which reduces the number of chosen inputs from the full space $\mathbb{F}_2^{n_b}$. Since the row reduction of $r(Z \mid v_i)$ can be precomputed, the operation times of the Gauss Elimination can be computed precisely instead of the maximum times $N \cdot (n_b + 1)$ for affine encoding and $N \cdot p$ for non-linear encoding.

Table 2: The parameters of CP-DCA, CP-LEA, and degree-2 CP-HDEA.

Attack	Partial Key Guess (Round Key Verification)				Partial Key Verification			
	$n_b = 4$				$n_b = 8$			
	c	N	z	Gauss	c	N	z	Gauss
CP-DCA	$\mathtt{0}$ ($\mathtt{1}$)	16	$\mathtt{0} - \mathtt{f}$	-	$\mathtt{0}$	256	$\mathtt{00} - \mathtt{ff}$	-
CP-LEA		6	$\mathtt{0, 8, 4, 2, 1, f}$	9		10	$\mathtt{0, 80, 40, 20, 10}$ $\mathtt{8, 4, 2, 1, ff}$	17
Degree-2 CP-HDEA		12	$\mathtt{0, 8, 4, 2, 1, c}$ $\mathtt{a, 9, 6, 5, 3, f}$	34		38	$\mathtt{0, 80, 40, 20, 10}$ $\mathtt{8, 4, 2, 1, c0}$ $\mathtt{a0, 90, 88, 84, 82}$ $\mathtt{81, 60, 50, 48, 44}$ $\mathtt{42, 41, 30, 28, 24}$ $\mathtt{22, 21, 18, 14, 12}$ $\mathtt{11, c, a, 9, 6, 5, 3, ff}$	151

To verify our theoretical analysis, we perform the simulations of CP-DCA, CP-LEA, and CP-HDEA against the 32/64/128-bit encoded function F'_{k^*} (refer to Figure 5) with affine output encodings. The random encodings are generated by the optimized white-box matrix library `WBMatrix` [TGS+22]. Table 3 depicts the attack results. Three attack phases of CP-CA have the maximum number $|\mathcal{K}|_{\mathtt{max}}$ and the minimum number $|\mathcal{K}|_{\mathtt{min}}$ of key guesses. Consequently, the round key verification phase can reduce the number of key candidates to 1, which indicates a successful key recovery. The symbol $t_{\mathtt{min}}$ refers to the minimum length of computation traces. It implies that CP-CA can extract the correct key with at least $t_{\mathtt{min}}$-bit intermediate leakage. Compared with CP-DCA, CP-LEA requires fewer samples in the traces to distinguish the correct key. Based on $t_{\mathtt{min}}$, n_b, $|\mathcal{K}|_{\mathtt{max}}$, N, and the operation times of Gauss Elimination, we can precisely compute the time complexity of the attack phases. The time complexity in Table 3 refers to the maximum complexity among the three phases, which can be represented as the overall complexity of the corresponding CP-CA attack. The time complexity of CP-DCA depends on the partial key verification phase while the partial key guess phase of CP-LEA has the highest time complexity. The experimental results reveal that the CP-DCA against the simulated target functions with the block sizes from 32 to 128 requires the time complexities from $2^{25.58}$ to $2^{31.70}$. The time complexities of CP-LEA against the simulated target functions with the block sizes from 32 to 128 are from $2^{12.17}$ to $2^{15.75}$.

Table 3: Simulated CP-DCA and CP-LEA against affine encoding.

Attack	Block Size	$t_{\mathtt{min}}$	Partial Key				Round Key Verification		Time Complexity
			Guess		Verification				
			$\lvert\mathcal{K}\rvert_{\mathtt{max}}$	$\lvert\mathcal{K}\rvert_{\mathtt{min}}$	$\lvert\mathcal{K}\rvert_{\mathtt{max}}$	$\lvert\mathcal{K}\rvert_{\mathtt{min}}$	$\lvert\mathcal{K}\rvert$	Recovery	
CP-DCA	32	32	48		12*		4		$2^{25.58}$
	64	64	80	16	20*	4	9	1	$2^{28.32}$
	128	128	208		52*		24		$2^{31.70}$
CP-LEA	32	4	32*						$2^{12.17}$
	64		64*	16	8	4	4	1	$2^{14.17}$
	128	6							$2^{15.75}$

* The attack phase corresponding to this number of key candidates has the highest time complexity.

6.2 Practical CP-CA Attacks

Implementations. The authors of self-equivalence [VRP22] and implicit white-box ARX ciphers [RVP22] provide the open-source codes to generate the white-box implementations of SPECK block cipher. We instantiate the SE-SPECK[2] with the sizes 32/K64, 48/K96, 64/K128, 96/K144, and 128/K256. Specifically, the scripts of IF-SPECK[3] only support for the sizes 32/K64, 64/K128, and 128/K256. Since the compilation of IF-SPECK128/K256 requires a significant amount of time, we only instantiate IF-SPECK32/K64 and IF-SPECK64/K128. Particularly, IF-SPECK can be implemented with affine or affine-quadratic self-equivalence. Thus, it can be constructed with quadratic, cubic, and quartic implicit round functions. Table 4 illustrates the performance of the implementations. We note that the degree in the table refers to the degree of the implicit round function. The abbreviation SE32 (resp. IF32) represents SE-SPECK32 (resp. IF-SPECK32) while K64 refers to a key size 64. The results are measured with an Intel Core i7-11800H processor @2.30GHz and 40GB RAM. The results reveal that IF-SPECK requires more computational resources than SE-SPECK. Moreover, a higher degree of the implicit function will increase memory and time costs.

Attack Results. To validate the CP-CA, we mount the CP-DCA, CP-LEA, and CP-HDEA attacks against SE-SPECK and IF-SPECK. Because of the impractical time complexity of CP-HDCA on breaking the quadratic encoding, we only mount the CP-DCA on attacking the affine encoded implementations. Table 5 depicts the practical attack results. The abbreviations D2, D3, and D4 represent the degrees 2, 3, and 4 of the implicit round functions in IF-SPECK respectively. The symbol $t_{\mathtt{min}}$ represents the minimum length of computation traces while $|k|_{\mathtt{max}}$ and $|k|_{\mathtt{min}}$ refer to the maximum and minimum numbers of the key guesses, respectively. Particularly, since the modular addition has sparse affine self-equivalences, CP-DCA cannot directly recover the secret key of SE-SPECK. By applying an extra random affine encoding to the output traces, CP-DCA can successfully extract the secret key. The overall time complexity of the round key recovery of CP-DCA and CP-HDEA is related to the partial key verification phase while the one of CP-LEA

[2]`https://github.com/jvdsn/white-box-speck`
[3]`https://github.com/ranea/whiteboxarx`

Table 4: Performance of SE-SPECK and IF-SPECK.

Cipher	Encoding	Degree	Source Code Size (MB)	Binary Size (MB)	RAM (MB)	Execution Time (ms)
SE32/K64	affine	-	0.08	0.04	1.11	0.01
SE48/K96			0.18	0.07	1.17	0.03
SE64/K128			0.35	0.13	1.27	0.05
SE96/K144			0.83	0.29	1.35	0.15
SE128/K256			1.70	0.57	1.74	0.23
IF32/K64	affine	2	0.16	0.15	3.08	2.43
	quadratic	3	1.85	1.82	5.30	11.63
		4	17.45	17.41	24.43	83.33
IF64/K128	affine	2	1.26	1.26	4.70	18.87
	quadratic	3	35.44	35.12	45.10	166.67
		4	691.11	690.09	788.20	$2,390.00$

depends on the partial key guess phase. The results demonstrate that SE-SPECK and IF-SPECK are vulnerable to the CP-CA attacks. The time complexities of CP-DCA against SE-SPECK with the block sizes from 32 to 128 are from $2^{25.58}$ to $2^{31.70}$ while defeating IF-SPECK32/64 with affine encoding has the time complexities $2^{25.58}$ and $2^{28.32}$. The time complexities of CP-LEA against SE-SPECK with the block sizes from 32 to 128 are from $2^{14.17}$ to $2^{18.17}$ while CP-LEA can break affine encoded IF-SPECK32/64 with the time complexities $2^{13.17}$ and $2^{14.17}$. The IF-SPECK32/64 with quadratic self-equivalence can be defeated by the degree-2 CP-HDEA with the time complexities from $2^{16.24}$ to $2^{18.24}$.

6.3 Comparisons

Comparison between CP-CA and Adaptive SCA. CP-CA focuses on the computation analysis between the chosen inputs (also the chosen intermediate values) and the traces. It is different from the adaptive SCA [KB07, VS10], which applies the modeling analysis related to the plaintexts. Without loss of generality, our proposed CP-CA is independent of the existing adaptive SCA techniques. Specifically, the reverse function in CP-CA seems similar to the adaptive chosen-plaintext phase in the *adaptive side-channel analysis* (ASCA) [TGS+21]. However, the procedures and the principles of the two attacks are substantially different. Figure 13 illustrates the flowcharts of CP-CA and ASCA. The detailed differences are compared as follows.

- Different procedures. CP-CA first constructs a reverse function and obtains the required plaintexts by choosing the inputs to the reverse function. CP-CA then mounts the computation analysis on the collected traces with the obtained plaintexts. Differently, ASCA first collects and analyzes some intermediate values by invoking the cryptographic algorithm with random plaintexts. Subsequently, ASCA chooses the computed plaintexts and encrypts them again to mount an SCA attack on the cryptographic algorithm.

- Different principals. CP-CA constructs a reverse function to cancel the first few round functions of the cryptographic algorithm. CP-CA aims to reduce the complexity of the enumeration on the large affine/non-linear encoding. Diversely, ASCA chooses some plaintexts by analyzing the intermediate values of the cryptographic algorithm. It intends to bypass some countermeasures such as masking and redundancy against the SCA with some properties of the chosen plaintexts.

Table 5: Practical CP-DCA, CP-LEA, and CP-HDEA against SE-SPECK and IF-SPECK.

Attack	Cipher	$t_{\mathtt{min}}$	Partial Key				Round Key		Time Complexity
			Guess		Verification		Verification		
			$\lvert\mathcal{K}\rvert_{\mathtt{max}}$	$\lvert\mathcal{K}\rvert_{\mathtt{min}}$	$\lvert\mathcal{K}\rvert_{\mathtt{max}}$	$\lvert\mathcal{K}\rvert_{\mathtt{min}}$	$\lvert\mathcal{K}\rvert$	Recovery	
CP-DCA	SE32/K64*	32	48	16	12**	4	4	1	$2^{25.58}$
	SE48/K96*	48					5		$2^{26.75}$
	SE64/K128*	64	80		20**		9		$2^{28.32}$
	SE96/K144*	96							$2^{29.49}$
	SE128/K256*	128	208		52**		24		$2^{31.70}$
	IF32/K64/D2	32	48		12**		4		$2^{25.58}$
	IF64/K128/D2	64	80		20**	3	9		$2^{28.32}$
CP-LEA	SE32/K64	16	32**	16	8	4	4	1	$2^{14.17}$
	SE48/K96	24	64**						$2^{16.34}$
	SE64/K128	32							$2^{17.17}$
	SE96/K144	48							$2^{18.34}$
	SE128/K256	64	32**						$2^{18.17}$
	IF32/K64/D2	8							$2^{13.17}$
	IF64/K128/D2								$2^{14.17}$
CP-HDEA	IF32/K64/D3	8	64	16	32**	16	8	1	$2^{16.24}$
	IF32/K64/D4								
	IF64/K128/D3								$2^{17.24}$
	IF64/K128/D4		128		64**				$2^{18.24}$

* The collected computation traces are encoded by an extra random affine mapping.
** The attack phase corresponding to this number of key candidates has the highest time complexity.

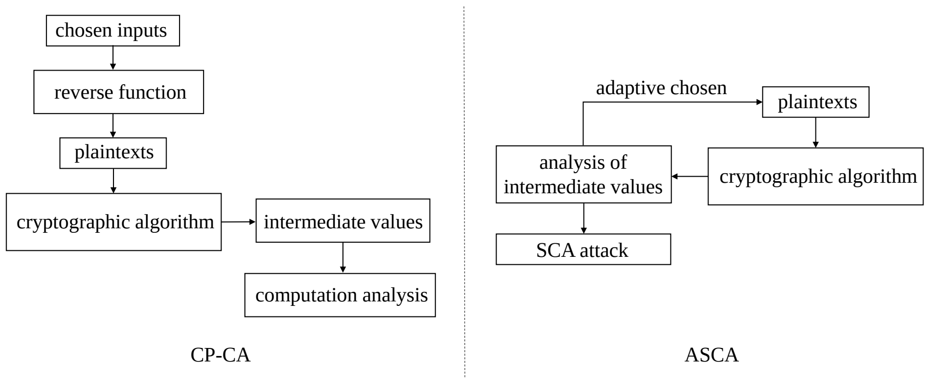

Figure 13: The comparison between CP-CA and ASCA.

Comparison between CP-LEA and ADCA. Although CP-LEA is a reformulation of ADCA, there are also some differences between CP-LEA and ADCA as follows.

- New reverse function. ADCA targets the encoded LUTs for Sbox-based white-box ciphers in the CEJO framework. It computes the inputs $x' = S^{-1}(x) + k$ (in byte level) for a key guess k. Differently, CP-LEA aims to attack the ARX-based white-box ciphers with large encoding and modular addition structures. It constructs a reverse function (in block size) by some reverse steps of the algorithm to choose the plaintexts.
- New chosen-plaintext attack. We note that CP-LEA has a chosen-plaintext process. The chosen plaintexts help to reduce the large space of input and encoding, which implies a feasible computation analysis on ARX-based white-box ciphers. On the contrary, ADCA does not require any particular plaintexts.
- Partial key guess. CP-LEA has the partial key guess phase to reduce the space of key guesses while ADCA needs to enumerate the key candidates over the key space.
- No algebraic degree detection. CP-LEA focuses on the linear mappings from the chosen intermediate values to a part of the traces. It distinguishes the correct key with the most solvable linear systems. Different from CP-LEA, ADCA detects the algebraic degrees of the computed mapping and counts the number of degrees that are no more than a specific value d.

Comparison between CP-LEA and LDA. By exploiting the technique of LDA, the algebraic attack [BLU23] demonstrated the vulnerability of IF-SPECK. The key difference between CP-LEA and LDA is that CP-LEA targets the linear mapping from the unencoded intermediates to a sample of traces, while LDA maps the encoded output to the unencoded intermediates reversely. Hence, CP-LEA can analyze the bits in the traces separately while LDA must locate the whole output bits. The detailed differences between CP-LEA and LDA against SE-SPECK and IF-SPECK are described as follows. Figure 14 also illustrates these differences.

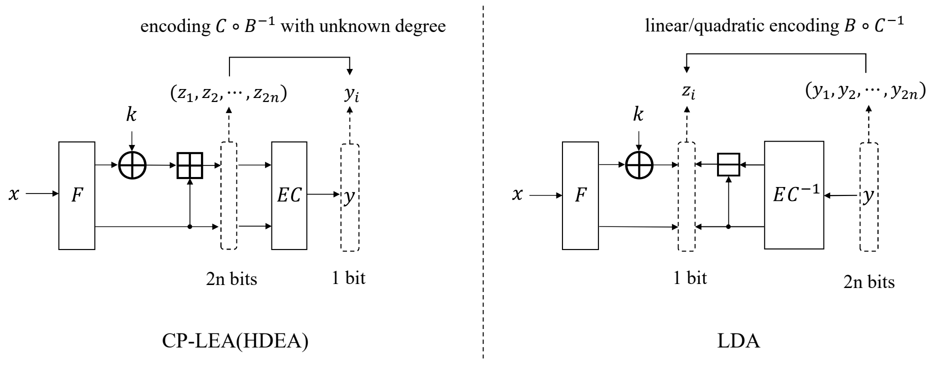
encoding $C \circ B^{-1}$ with unknown degree
$(z_1, z_2, \cdots, z_{2n})$
y_i
k
x
F
EC
y
2n bits
1 bit
CP-LEA(HDEA)
linear/quadratic encoding $B \circ C^{-1}$
z_i
$(y_1, y_2, \cdots, y_{2n})$
k
x
F
EC^{-1}
y
1 bit
2n bits
LDA

Figure 14: The comparison between CP-LEA and LDA.

- Different target functions. LDA focuses on the mapping from the outputs of the target function to its intermediate values. This mapping is the inverse encoding (i.e., decoding) $EC^{-1} = B \circ C^{-1}$ of which the degree depends on the affine/quadratic encoding B. It constructs a linear/quadratic system between the $2n$-bit encoded output $y = (y_1, y_2, \cdots, y_{2n})$ and an intermediate value $z_i \in \mathbb{F}_2$. The sensitive value z_i is calculated through the public function F and some guessed key bits k from the input x. For the correct key, a unique mapping will be determined from y to z_i. Diversely, CP-LEA focuses on the mapping from the intermediate values of the target function to output traces. This mapping is the encoding $EC = C \circ B^{-1}$ of which the degree is unknown when the quadratic encoding B is applied. It calculates the $2n$-bit $z = (z_1, z_2, \cdots, z_{2n})$ of the modular addition based on a partial key guess k. Subsequently, it focuses on the (reduced) linear mapping from z to an encoded output bit y_i. CP-LEA(HDEA) computes a (higher-degree extended) linear system between the chosen input z and the output bit y_i to distinguish the correct key.

- Different windows. LDA needs to locate a w-size ($w \geq 2n$) window of the intermediate values which contain all the bits of the encoded output y. It cannot recover the key with limited access to the encoded output y. Contrarily, CP-LEA does not require full access to y. It focuses on a t-size ($t \leq w$) window of y, which implies that CP-LEA still can recover the correct key when analyzing the partial bits of y.
- Different round key candidates. Based on the attack results of the algebraic attack [BLU23], LDA cannot uniquely recover the round key of IF-SPECK. It obtains 4 candidates for a round and needs to determine the correctness in the next-round key recovery. Thus, it is required to iterate the attack over 5 rounds for the master key recovery. Differently, CP-LEA has three attack phases among which the round key verification phase can uniquely distinguish the correct key. Hence, CP-LEA can extract the master key by iterating the attack over 4 rounds instead of 5 rounds.

6.4 Different Constructions and Possible Countermeasures

CP-CA performs a modeling analysis on the computation traces which are the encoded sensitive values of the cryptographic algorithm. It focuses on the round outputs instead of the polynomials of the implicit function. The different constructions of an implicit function, such as the generation of $P(x, y) = 0$ cannot enhance the security against CP-CA. For SE-SPECK, it is vulnerable to the algebraic attack which leverages some properties of the sparse matrix when generating the affine self-equivalences. Following the gray-box attack context, CP-CA has no access to the generation of white-box implementations. The sparse affine self-equivalence of SE-SPECK has been abstracted as a random affine encoding. Thus, it is a more general case for SE-SPECK. A specific computation analysis for considering the properties of the sparse affine self-equivalence is left as future work.

Similar to ADCA, CP-HDEA is a higher-degree computation analysis against the non-linear self-equivalence in the implicit function framework. By fixing most parts of the inputs as constant, a higher-degree CP-HDEA enjoys a lower time complexity than LDA. For any higher-degree non-linear encoding, the attack degree d of CP-HDEA satisfies $d \leq n_b$, where n_b is the size of chosen inputs ($n_b = 4/8$ in practice). Hence, even higher-degree self-equivalences might be still vulnerable to higher-degree CP-HDEA. Since the implicit function represents the round function as multivariate binary polynomials, it is possible to construct a multi-round implicit function. For a multi-round key recovery, CP-CA needs to search the round keys in a larger space with a higher time complexity. Hence, the multi-round implicit function might be a possible countermeasure to resist CP-CA. However, it is still challenging to represent the multi-round functions as a low-degree implicit function thus we left it for future work.

6.5 Application to CRAX Block Cipher

CRAX is a lightweight 64-bit block cipher that applies a 64-bit ARX-based Sbox `Alzette` [BBdS+20] and a 128-bit secret key. To explore the feasibility of CP-CA against another ARX cipher, we discuss an attack on the self-equivalence (SE-CRAX) and implicit white-box CRAX (IF-CRAX) implementations. Figure 15 depicts the structures of the reverse function

G_{k_0,k_1} and the encoded function $F'_{k_0^*,k_1^*}$, where $(k_0,k_1) \in (\mathbb{F}_2^{32})^2$ and $(k_0^*,k_1^*) \in (\mathbb{F}_2^{32})^2$ denote the key guess and the correct key, respectively.

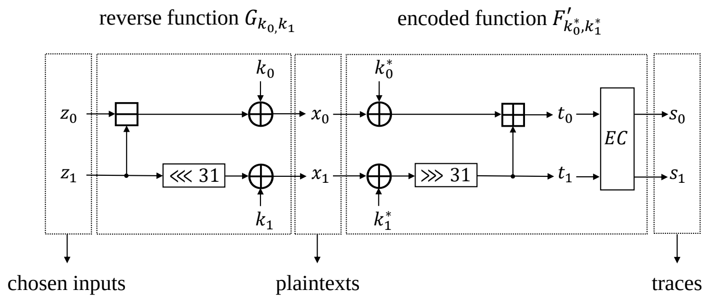

Figure 15: The reverse function of CP-CA against SE-CRAX and IF-CRAX.

With the composition of $F_{k_0^*,k_1^*}$ and G_{k_0,k_1}, the sensitive values (t_0,t_1) can be derived from the inputs (z_0,z_1) as follows.

$$F_{k_0^*,k_1^*} \circ G_{k_0,k_1} : (\mathbb{F}_2^{32})^2 \mapsto (\mathbb{F}_2^{32})^2 : F_{k_0^*,k_1^*} \circ G_{k_0,k_1}(z_0,z_1) = (t_0,t_1)$$
$$\begin{aligned} t_1 &= ((z_1 \lll 31) \oplus k_1 \oplus k_1^*) \ggg 31 \\ &= z_1 \oplus (\Delta k_1 \ggg 31) \qquad (3) \\ t_0 &= ((z_0 \boxminus z_1) \oplus k_0 \oplus k_0^*) \boxplus t_1 \\ &= ((z_0 \boxminus z_1) \oplus \Delta k_0) \boxplus (z_1 \oplus (\Delta k_1 \ggg 31)) \qquad (4) \end{aligned}$$

For a correct key guess, the variables in Eq. (3) and Eq. (4) can be canceled out as

$$t_1 = z_1, \;\; t_0 = (z_0 \boxminus z_1) \boxplus z_1 = z_0.$$

The unencoded intermediate values (t_0,t_1) are equal to the inputs (z_0,z_1). For an incorrect key guess, the unencoded sensitive values (t_0,t_1) are computed by Eq. (3) and Eq. (4). It is a random function for different key candidates $(k_0,k_1) \neq (k_0^*,k_1^*)$.

Partial Key Guess. Similar to the attack process on breaking SE-SPECK and IF-SPECK, CP-CA guesses a nibble key ($2^4 = 16$ key candidates) and constructs the corresponding reverse function to distinguish the possible correct key. Since the round function of CRAX contains two 32-bit subkeys, CP-CA needs to recover the k_0 and k_1 separately. For extracting k_0, CP-CA fixes z_0 as 0 and removes the subtraction $z_0 \boxminus z_1$, such that Eq. (3) and Eq. (4) can be transformed as

$$\begin{aligned} t_1 &= z_1 \oplus (\Delta k_1 \ggg 31), \\ t_0 &= \Delta k_0 \boxplus (z_1 \oplus (\Delta k_1 \ggg 31)) \, . \end{aligned}$$

CP-CA fixes the key guess k_1 as a constant and focuses on the partial key enumeration on k_0. There is a sliding window with nibble size in both the key guess k_0 and the chosen input z_1. Therefore, it recovers the partial key from the least significant nibble to the most significant one. For recovering k_1, CP-CA fixes z_1 as 0, such that Eq. (3) and Eq. (4) can be transformed as

$$\begin{aligned} t_1 &= \Delta k_1 \ggg 31, \\ t_0 &= (z_0 \oplus \Delta k_0) \boxplus (\Delta k_1 \ggg 31). \end{aligned}$$

CP-CA fixes the key guess k_0 as a constant and focuses on the partial key enumeration on k_1. By applying the computation analysis between the chosen input z_0 and the traces, CP-CA can recover a set of possible correct key candidates of k_1. Specifically, the extracted key k_1 needs to be reset through a left circular shift by 31.

Partial Key Verification. This phase is equal to the attacks on SE-SPECK and IF-SPECK. It constructs a sliding attack window on the chosen input and the key guess with byte size to reduce the key candidates obtained from the partial key guess phase.

Round Key Verification. By obtaining the reduced key candidates of k_0 and k_1, CP-CA fixes the input z_1 as a non-zero constant value and chooses the nibble input z_0 to further distinguish the correct key among $\{k_0, k_1\}$.

Attack Results. We implement SE-CRAX and degree-2/3 IF-CRAX by modifying the open-source code of SE-SPECK and IF-SPECK. Table 6 illustrates the attack results of CP-DCA, CP-LEA, and CP-HDEA against SE-CRAX and IF-CRAX. The partial key verification phase of CP-DCA on SE-CRAX returns 9 and 5 candidates for k_0 and k_1, respectively while attacking IF-CRAX recovers 17 and 5 candidates for k_0 and k_1, respectively. The same phase of CP-LEA on SE-CRAX and IF-CRAX/D2 computes 8 and 4 candidates for k_0 and k_1, respectively while CP-HDEA against IF-CRAX/D3 obtains 32 and 8 candidates for k_0 and k_1, respectively. The round key verification phases of CP-DCA and CP-LEA/HDEA finally reduce the 45, 85, 32, and 256 key candidates to 8 ones. Similar to the algebraic attack on IF-SPECK, CP-CA on SE-CRAX and IF-CRAX leaves with 8 candidates which contain the correct one after the round key recovery. The master key will be recovered by iterating the attack over 5 rounds.

Table 6: Practical CP-DCA, CP-LEA, and CP-HDEA against SE-CRAX and IF-CRAX.

Attack	Cipher	$t_{\texttt{min}}$	Partial Key				Round Key Verification				Time Complexity
			Guess		Verification						
			$\lvert\mathcal{K}\rvert_{\texttt{max}}$	$\lvert\mathcal{K}\rvert_{\texttt{min}}$	$\lvert\mathcal{K}\rvert_{\texttt{max}}$	$\lvert\mathcal{K}\rvert_{\texttt{min}}$	$\lvert k_0\rvert$	$\lvert k_1\rvert$	$\lvert\mathcal{K}\rvert$	Recovery	
CP-DCA	SE-CRAX*	64	48	16	25**	6	9	5	45	8	$2^{28.64}$
	IF-CRAX/D2	64	80		41**		17	5	85		$2^{29.36}$
CP-LEA	SE-CRAX	32	32**	16	16	4	8	4	32	8	$2^{16.17}$
	IF-CRAX/D2	12									$2^{14.75}$
CP-HDEA	IF-CRAX/D3	9	64	16	64**	16	32	8	256	8	$2^{18.41}$

* The collected computation traces are encoded by an extra random affine mapping.
** The attack phase corresponding to this number of key candidates has the highest time complexity.

To the best of our knowledge, there are no other publicly available ARX-based white-box ciphers except for the SPECK block cipher. It is an elaborate work to apply the self-equivalence and implicit function frameworks to other ARX ciphers, such as LEA [HLK+13], SPARX [DPU+16], and CHAM [KRK+18]. Specifically, the key problem is to cancel the encodings when the round function has more than two branches and to merge the self-equivalences when the round function consists of multiple modular additions. The constructions and the CP-CA attacks of the other ARX-based white-box ciphers are left as future work.

7 Conclusion

This paper studied the computation analysis against ARX-based white-box ciphers. We first analyzed the encoded structure of SE-SPECK and IF-SPECK. Subsequently, its resistance to DCA and ADCA attacks has been discussed. The analysis reveals that the large spaces of input, key candidate, and encoding can practically mitigate DCA attacks. To reduce the complexity, we introduced the concept of a reverse function and proposed a new CP-CA attack method with two instances of CP-DCA and CP-LEA. CP-CA can reduce the large affine/non-linear encoding into a small one by elaborately choosing the intermediate values. The theoretical and experimental results demonstrate that both self-equivalence and implicit SPECK implementations are vulnerable to the CP-DCA and CP-LEA attacks.

This is the first attempt to evaluate the security of SE-SPECK and IF-SPECK against computation analysis. It is left as future work to explore the possibility of constructing a multi-round implicit function as a countermeasure against CP-CA. Moreover, it is also a challenge to evaluate the resistances of other ARX-based white-box ciphers with multiple branches and modular additions to the CP-CA attacks.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable comments. We are also grateful to Adrián Ranea and Jian Guo for their insightful suggestions on a previous version of this paper. This work was supported by the National Natural Science Foundation of China (62132008, U2336209, 62072192, 62071222, U22B2030) and Guangdong Basic and Applied Basic Research Foundation (2022A1515140090). Zhe Liu is also supported by the Natural Science Foundation of Jiangsu Province (BK20220075) and the Fok Ying-Tong Education Foundation for Young Teachers in the Higher Education Institutions of China (No.20193218210004).

References

[BBB$^+$19] Estuardo Alpirez Bock, Joppe W. Bos, Chris Brzuska, Charles Hubain, Wil Michiels, Cristofaro Mune, Eloi Sanfelix Gonzalez, Philippe Teuwen, and Alexander Treff. White-box cryptography: Don't forget about grey-box attacks. *J. Cryptol.*, 32(4):1095–1143, 2019.

[BBdS$^+$20] Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, and Qingju Wang. Alzette: A 64-bit arx-box - (feat. CRAX and TRAX). In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 419–448. Springer, 2020.

[BBMT18] Estuardo Alpirez Bock, Chris Brzuska, Wil Michiels, and Alexander Treff. On the ineffectiveness of internal encodings - revisiting the DCA attack on white-box cryptography. In Bart Preneel and Frederik Vercauteren, editors, *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*, volume 10892 of *Lecture Notes in Computer Science*, pages 103–120. Springer, 2018.

[BCBP03] Alex Biryukov, Christophe De Cannière, An Braeken, and Bart Preneel. A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 33–50. Springer, 2003.

[BCD06] Julien Bringer, Hervé Chabanne, and Emmanuelle Dottax. White box cryptography: Another attempt. *IACR Cryptol. ePrint Arch.*, page 468, 2006.

[BCH16] Chung Hun Baek, Jung Hee Cheon, and Hyunsook Hong. White-box AES implementation revisited. *J. Commun. Networks*, 18(3):273–287, 2016.

[BGE04] Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Cryptanalysis of a white box AES implementation. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers*, volume 3357 of *Lecture Notes in Computer Science*, pages 227–240. Springer, 2004.

[BHMT16] Joppe W. Bos, Charles Hubain, Wil Michiels, and Philippe Teuwen. Differential computation analysis: Hiding your white-box designs is not enough. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 215–236. Springer, 2016.

[BLU23] Alex Biryukov, Baptiste Lambin, and Aleksei Udovenko. Cryptanalysis of arx-based white-box implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(3):97–135, 2023.

[BSS$^+$13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The simon and speck families of lightweight block ciphers. Cryptology ePrint Archive, Paper 2013/404, 2013.

[BT20] Estuardo Alpirez Bock and Alexander Treff. Security assessment of white-box design submissions of the CHES 2017 CTF challenge. In Guido Marco Bertoni and Francesco Regazzoni, editors, *Constructive Side-Channel Analysis and Secure Design - 11th International Workshop, COSADE 2020, Lugano, Switzerland, April 1-3, 2020, Revised Selected Papers*, volume 12244 of *Lecture Notes in Computer Science*, pages 123–146. Springer, 2020.

[BU18] Alex Biryukov and Aleksei Udovenko. Attacks and countermeasures for white-box designs. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 373–402. Springer, 2018.

[CEJvO02a] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. White-box cryptography and an AES implementation. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002. Revised Papers*, volume 2595 of *Lecture Notes in Computer Science*, pages 250–270. Springer, 2002.

[CEJvO02b] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. A white-box DES implementation for DRM applications. In Joan Feigenbaum, editor, *Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002, Washington, DC, USA, November 18, 2002, Revised Papers*, volume 2696 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002.

[CGM21] Claude Carlet, Sylvain Guilley, and Sihem Mesnager. Structural attack (and repair) of diffused-input-blocked-output white-box cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):57–87, 2021.

[DFLM18] Patrick Derbez, Pierre-Alain Fouque, Baptiste Lambin, and Brice Minaud. On recovering affine encodings in white-box implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):121–149, 2018.

[DPU+16] Daniel Dinu, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, Johann Großschädl, and Alex Biryukov. Design strategies for arx with provable bounds: Sparx and lax. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 484–513, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[GPRW20] Louis Goubin, Pascal Paillier, Matthieu Rivain, and Junwei Wang. How to reveal the secrets of an obscure white-box implementation. *J. Cryptogr. Eng.*, 10(1):49–66, 2020.

[HLK+13] Deukjo Hong, Jung-Keun Lee, Dong-Chan Kim, Daesung Kwon, Kwon Ho Ryu, and Donggeon Lee. LEA: A 128-bit block cipher for fast encryption on common processors. In Yongdae Kim, Heejo Lee, and Adrian Perrig, editors, *Information Security Applications - 14th International Workshop, WISA 2013, Jeju Island, Korea, August 19-21, 2013, Revised Selected Papers*, volume 8267 of *Lecture Notes in Computer Science*, pages 3–27. Springer, 2013.

[Kar10] Mohamed Karroumi. Protecting white-box AES with dual ciphers. In Kyung Hyune Rhee and DaeHun Nyang, editors, *Information Security and Cryptology - ICISC 2010 - 13th International Conference, Seoul, Korea, December 1-3, 2010, Revised Selected Papers*, volume 6829 of *Lecture Notes in Computer Science*, pages 278–291. Springer, 2010.

[KB07] Boris Köpf and David A. Basin. An information-theoretic model for adaptive side-channel attacks. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 286–296. ACM, 2007.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KRK+18] Bonwook Koo, Dongyoung Roh, Hyeonjin Kim, Younghoon Jung, Dong-Geon Lee, and Daesung Kwon. Cham: A family of lightweight block ciphers for resource-constrained devices. In Howon Kim and Dong-Chan Kim, editors, *Information Security and Cryptology – ICISC 2017*, pages 3–25, Cham, 2018. Springer International Publishing.

[LN05] Hamilton E. Link and William D. Neumann. Clarifying obfuscation: Improving the security of white-box DES. In *International Symposium on Information Technology: Coding and Computing (ITCC 2005), Volume 1, 4-6 April 2005, Las Vegas, Nevada, USA*, pages 679–684. IEEE Computer Society, 2005.

[LRM+13] Tancrède Lepoint, Matthieu Rivain, Yoni De Mulder, Peter Roelse, and Bart Preneel. Two attacks on a white-box AES implementation. In Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, volume 8282 of *Lecture Notes in Computer Science*, pages 265–285. Springer, 2013.

[MS16] Brendan McMillion and Nick Sullivan. Attacking white-box AES constructions. In Brecht Wyseur and Bjorn De Sutter, editors, *Proceedings of the 2016 ACM Workshop on Software PROtection, SPRO@CCS 2016, Vienna, Austria, October 24-28, 2016*, pages 85–90. ACM, 2016.

[MWP10] Yoni De Mulder, Brecht Wyseur, and Bart Preneel. Cryptanalysis of a perturbated white-box AES implementation. In Guang Gong and Kishan Chand Gupta, editors, *Progress in Cryptology - INDOCRYPT 2010 - 11th International Conference on Cryptology in India, Hyderabad, India, December 12-15, 2010. Proceedings*, volume 6498 of *Lecture Notes in Computer Science*, pages 292–310. Springer, 2010.

[RP20] Adrián Ranea and Bart Preneel. On self-equivalence encodings in white-box implementations. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers*, volume 12804 of *Lecture Notes in Computer Science*, pages 639–669. Springer, 2020.

[RVP22] Adrián Ranea, Joachim Vandersmissen, and Bart Preneel. Implicit white-box implementations: White-boxing ARX ciphers. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part I*, volume 13507 of *Lecture Notes in Computer Science*, pages 33–63. Springer, 2022.

[RW19] Matthieu Rivain and Junwei Wang. Analysis and improvement of differential computation attacks against internally-encoded white-box implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):225–255, 2019.

[SMG16] Pascal Sasdrich, Amir Moradi, and Tim Güneysu. White-box cryptography in the gray box - - A hardware implementation and its side channels -. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 185–203. Springer, 2016.

[TGCX23] Yufeng Tang, Zheng Gong, Jinhai Chen, and Nanjiang Xie. Higher-order DCA attacks on white-box implementations with masking and shuffling countermeasures. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(1):369–400, 2023.

[TGLZ23] Yufeng Tang, Zheng Gong, Bin Li, and Liangju Zhao. Revisiting the computation analysis against internal encodings in white-box implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(4):493–522, 2023.

[TGS$^+$21] Yufeng Tang, Zheng Gong, Tao Sun, Jinhai Chen, and Fan Zhang. Adaptive side-channel analysis model and its applications to white-box block cipher implementations. In Yu Yu and Moti Yung, editors, *Information Security and Cryptology - 17th International Conference, Inscrypt 2021, Virtual Event, August 12-14, 2021, Revised Selected Papers*, volume 13007 of *Lecture Notes in Computer Science*, pages 399–417. Springer, 2021.

[TGS$^+$22] Yufeng Tang, Zheng Gong, Tao Sun, Jinhai Chen, and Zhe Liu. WBMatrix: An optimized matrix library for white-box block cipher implementations. *IEEE Trans. Computers*, 71(12):3375–3388, 2022.

[VRP22] Joachim Vandersmissen, Adrián Ranea, and Bart Preneel. A white-box speck implementation using self-equivalence encodings. In Giuseppe Ateniese and Daniele Venturi, editors, *Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings*, volume 13269 of *Lecture Notes in Computer Science*, pages 771–791. Springer, 2022.

[VS10] Nicolas Veyrat-Charvillon and François-Xavier Standaert. Adaptive chosen-message side-channel attacks. In Jianying Zhou and Moti Yung, editors, *Applied Cryptography and Network Security, 8th International Conference, ACNS 2010, Beijing, China, June 22-25, 2010. Proceedings*, volume 6123 of *Lecture Notes in Computer Science*, pages 186–199, 2010.

[WMGP07] Brecht Wyseur, Wil Michiels, Paul Gorissen, and Bart Preneel. Cryptanalysis of white-box DES implementations with arbitrary external encodings. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers*, volume 4876 of *Lecture Notes in Computer Science*, pages 264–277. Springer, 2007.