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

[2]Zhejiang Lab

ASK 2023, JNU, Guangzhou

Outline

1. ARX-Based White-Box Ciphers
2. Computation Analysis against White-Box SPECK
3. Chosen-Plaintext Computation Analysis
4. Attack Instances of CP-DCA and CP-ADCA
5. Conclusion

Outline

White-Box Cryptography

- Goal: prevents the cryptographic algorithm from the key extraction in white-box context.
- Technique: applies the encoding to hide the sensitive information.
- Framework:
  - CEJO (SAC'02): a network of look-up tables (LUTs) with linear/non-linear encodings.
  - Self-equivalence (SAC'20): hides the key in the affine layer with self-equivalence encodings.
  - Implicit function (CRYPTO'22): hides the key in the binary multivariate polynomials with self-equivalence and linear/non-linear encodings.

LUTs in CEJO Framework

- At SAC 2002, Chow *et al.* proposed the CEJO framework.
- It transforms the round function into a series of look-up tables (LUTs) with embedded secret key and applies linear/non-linear encodings to protect the LUTs.

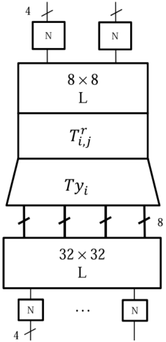


Figure: Type II: key-dependent T-boxes/ Ty_i table

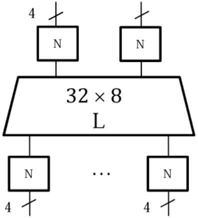


Figure: Type III: compatibility of encodings between consecutive rounds

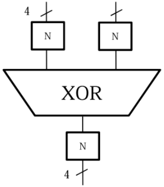


Figure: Type IV: encoded nibble XOR table

Self-Equivalence Encodings

- At SAC 2020, Ranea and Preneel proposed the self-equivalence framework.

Definition

(Self-equivalence). Let F be an n-bit function. A pair of n-bit affine permutations (A, B) such that $F = B \circ F \circ A$ is called an (affine) self-equivalence of F.

- Self-equivalence of Sbox: $S = B \circ S \circ A$.
- Round function: $E = L \circ S \circ \oplus k$.

$\Rightarrow$ Self-equivalence round function: $L \circ \underline{B \circ S \circ A} \circ \oplus k$.

Self-Equivalence White-Box Implementation

- Let $AL^r = A \circ \oplus k^r \circ L \circ B$.
- Round functions:

$$\bar{E}^n_{k^n} \circ \cdots \circ \bar{E}^1_{k^1} = (L \circ B) \circ S \circ (A \circ \oplus k^n \circ L \circ B) \circ S \circ \cdots \circ S \circ (A \circ \oplus k^1)$$

$$= (L \circ B) \circ S \circ AL^n \circ S \circ \cdots \circ S \circ AL^2 \circ S \circ (A \circ \oplus k^1).$$

- Introducing the external encodings P and Q such that:

$$AL^{n+1} = Q \circ L \circ B, \; AL^1 = A \circ \oplus k^1 \circ P.$$

- Encoded encryption: $\bar{E}_k = AL^{n+1} \circ S \circ \cdots \circ S \circ AL^1$

$\Rightarrow$ consists of the affine function AL and the substitution S.

Algebraic Attacks against CEJO and Self-Equivalence

- CEJO:
  - SASAS $\Rightarrow$ ASA $\Rightarrow$ affine equivalence problems.
  - CEJO-WBAES: time complexity 2^{22}, XL-WBAES: time complexity 2^{32}.
- Self-Equivalence:
  - Sparse matrix.
  - Diagonal encodings.
  - A few pairs of self-equivalences: 2040 of AES and SM4 Sbox.

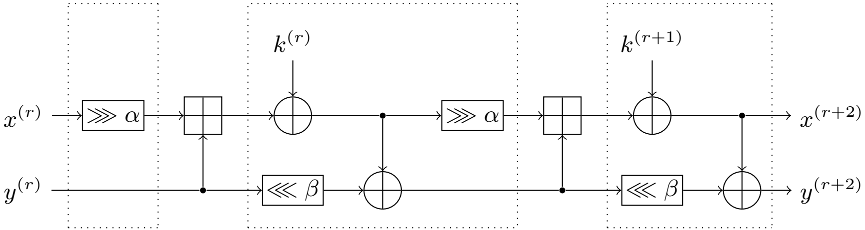
- Both frameworks apply small encodings to protect a small SBox.
- New direction: large substitution layer with many pairs of self-equivalences and large dimension of encodings.

Self-Equivalence White-Box SPECK

At ACNS 2022, Vandersmissen *et al.* proposed a self-equivalence white-box SPECK (SE-SPECK) implementations.

- Transforming the ARX to SPN:

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



Self-Equivalence White-Box SPECK

- Let $\overline{AL^{(r)}} = A \circ AL^{(r)} \circ B$, $S = B \circ S \circ A$,

$$E'_k = (AL^{(n)} \circ B) \circ S \circ (A \circ AL^{(r)} \circ B) \circ S \circ \cdots \circ S \circ AL^{(0)}$$

$$= (AL^{(n)} \circ B) \circ S \circ \overline{AL^{(n)}} \circ S \circ \cdots \circ S \circ \overline{AL^{(1)}} \circ S \circ AL^{(0)}.$$

- External encodings:
    - For some random bijections I and O:
    - $\overline{AL^{(0)}} = AL^{(0)} \circ I$.
    - $\overline{AL^{(n)}} = O \circ AL^{(n)} \circ B$.

Algebraic Attacks against SE-SPECK

- The key is embedded in the affine layer $AL^r = A \circ \oplus k^r \circ L \circ B$.
- The self-equivalences A and B has the sparse matrices with $2n + 11$ variable entries.
- The key can be recovered by constructing a system of linear equations based on a few of unknown variables.

Implicit Framework to ARX Ciphers

At CRYPTO 2022, Ranea *et al.* proposed an implicit framework to ARX Ciphers.

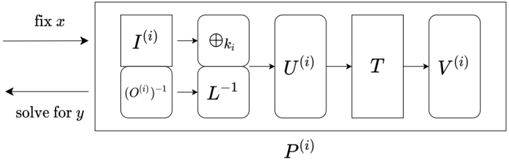
- Representing each round function by a low-degree implicit function (efficient implementation).
- Encoding the implicit function with large affine permutations and even large non-linear self-equivalences.
- Combining the large self-eqivalences with large affine encodings (hide the sparse matrix).

Implicit Framework to ARX Ciphers

Definition

(Implicit function). Let F be an n-bit function. A $(2n, m)$-bit function P is called an implicit function of F if it satisfies $P(x, y) = 0 \Leftrightarrow y = F(x)$.

- $I, O \leftarrow$ round encodings, $T \leftarrow$ implicit function of S, $U \leftarrow (B^{-1}, A)$ such that $S = B \circ S \circ A$, $V \leftarrow$ a random linear transformation.



- Implicit round function $\Rightarrow$ linear system:

$$P(x, y) = V \circ T \circ ((A \circ \oplus k \circ I(x), B^{-1} \circ L^{-1} \circ O^{-1}(y))) = 0$$

$$\Leftrightarrow \underline{V \circ B^{-1} \circ L^{-1} \circ O^{-1}}(y) = \underline{V \circ S \circ A \circ \oplus k \circ I}(x)$$

Implicit Framework to ARX Ciphers

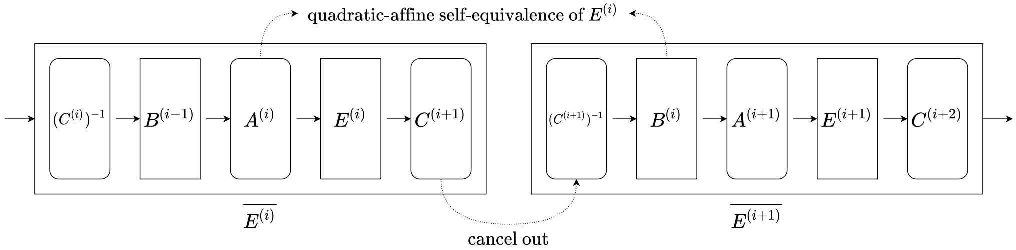
- Let $E = L \circ S \circ \bigoplus k$, without the representation of an implicit function:

$$\overline{E^{(i)}} = C^{(i+1)} \circ E^{(i)} \circ A^{(i)} \circ B^{(i-1)} \circ (C^{(i)})^{-1},$$

where

$$E^{(i)} = B^{(i)} \circ E^{(i)} \circ A^{(i)}.$$

- The canceling rule of the implicit self-equivalence implementation:



Algebraic Attacks against Implicit Framework

- The key is embedded in the implicit round function:

$$\overline{E^{(i)}} = C^{(i+1)} \circ (B^{(i)})^{-1} \circ E_k^{(i)} \circ B^{(i-1)} \circ (C^{(i)})^{-1}.$$

- The quadratic self-equivalences B is extremely sparse with a few monomials, up to affine equivalence.
- The key recovery on the structure *ASA* with modular addition S.
- Time complexity $\mathcal{O}(n^9)/\mathcal{O}(n^6)$ with/without external encodings [BLU23].

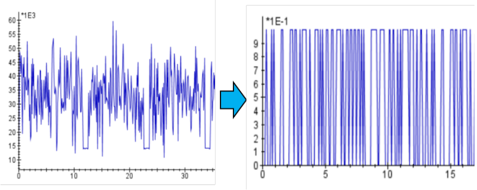
Outline

Differential Computation Analysis

At CHES 2016, differential computation analysis (DCA) was proposed to perform a statistical analysis on the computation traces of white-box implementations.

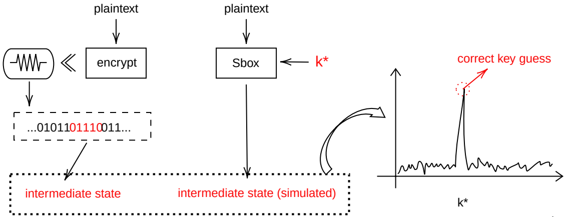
- **Noise-Free Computation Traces**

  The first/last-round computed values (accessed memory/register) by using DBI tools.



- **Differential Power Analysis**

  Dividing the traces in two distinct sets based on key guesses, computing the difference of two sets, distinguishing the correct key with the highest peak.



Differences between Algebraic Attack and DCA

Difference	Algebraic Attack	DCA
Context	white-box	gray-box
Process	1) unpacks the obfuscation layers 2) pinpoints the target function 3) decodes the encoded structure	1) collects the computation traces 2) analyzes the traces
Reverse Engineering	required	not required
Time Complexity	algebraic attack + reverse engineering (extra skills and time)	computation analysis

Computation Analysis (CA)

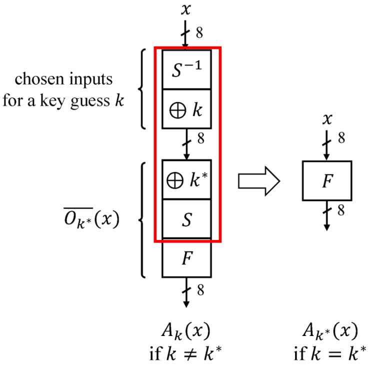
- Following DCA, many other computation analyses are proposed.
- $F = N \circ L \leftarrow$ the internal encoding, $\varphi \leftarrow$ a sensitive function.

Distinguisher	Attack Context	Method	Analysis of Key Leakage	Time Complexity
DCA [BHMT16]	gray-box	correlation computation	-	2^{22}
IDCA [BBB+19]	gray-box	correlation computation	$\mathtt{HW} = 1$ of L	2^{27}
CPA [RW19]	gray-box	correlation computation	non-injection of φ bijection of F	2^{22}
CA [RW19]	gray-box	correlation computation	non-injection of φ bijection of F	2^{29}
MIA [RW19]	gray-box	correlation computation	non-injection of φ bijection of F	2^{22}
SA [SMG16]	gray-box	spectral analysis	-	2^{27}
MSA [LJK20]	gray-box	spectral analysis	imbalance of L	2^{22}
ISA [CGM21]	white-box	spectral analysis	non-invertibility of L	2^{32}
ADCA [TGLZ23]	gray-box	detection of algebraic degree	$d_{alg}(F) \leq 6$	$2^{21.32} \sim 2^{24.07}$

Algebraic Degree Computation Analysis

At CHES 2023, Tang *et al.* proposed the algebraic degree computation analysis (ADCA).

- It can distinguish the correct key by computing the algebraic degrees of target functions.
- ADCA can break the most cases of encodings with the lowest time complexity.

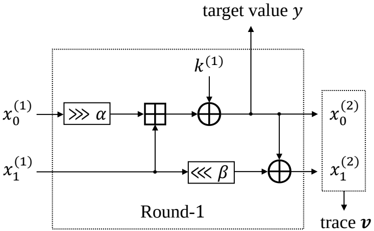


- $k^* \leftarrow$ secret key,
  Target function $O_{k^*} = F \circ S \circ \oplus k^*$.
- $k \leftarrow$ key guess,
  Chosen input $x' = \oplus k \circ S^{-1}(x)$.

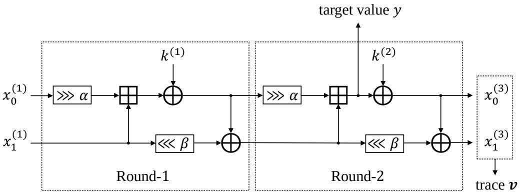
- If $k = k^*$, $A_{k^*} = F$ which is the internal encoding.
- If $k \neq k^*$, $A_k = F \circ S \circ \oplus k^* \circ \oplus k \circ S^{-1}$ which a random function.

Analysis of DCA against ARX Structure

- Targeting the first-round key addition.



- Targeting the second-round modular addition.



Simplified Target Functions

- The first-round key addition.

$$y = \left((x_0^{(1)} \ggg \alpha) \boxplus x_1^{(1)} \right) \oplus k$$

$$\Rightarrow y = (x \boxplus c_1) \oplus k$$
$$= S_1(x) \oplus k$$

- Key addition after substitution.

- The second-round modular addition.

$$y = \left(\left(\left((x_0^{(1)} \ggg \alpha) \boxplus x_1^{(1)} \right) \oplus k \right) \ggg \alpha \right) \boxplus \left((x_1 \lll \beta) \oplus \left((x_0^{(1)} \ggg \alpha) \boxplus x_1^{(1)} \right) \oplus k \right)$$

$$\Rightarrow y = ((x \boxplus c_1) \oplus k) \boxplus (c_2 \oplus (x \boxplus c_1) \oplus k)$$
$$= S_2(S_1(x) \oplus k, S_1(x) \oplus k \oplus c_2)$$

- Substitution after key addition.

DCA on SPECK

- Result: For both two target functions, DCA obtains the highest correlation 1 for every key guess and fails to distinguish the correct key.
- Reason:
  - The modular addition lacks confusion to fully obfuscate the key information.
  - The sensitive values of different key candidates are similar to each other.
  - At least one bit of the sensitive values for an incorrect key guess is equal to one bit of the sensitive values corresponding to the correct key.

Sum-Correlation DCA

- Principal:
  - Each bit of the sensitive values for the correct key has correlation 1.
  - The correct key has the maximum summed correlations.

- For each key guess, SC-DCA computes the correlation between each bit of the sensitive value $(\varphi_k(x))_i$ and each sample in the traces $\boldsymbol{v}_j$.
- It sums the maximum computed correlations of every bit in the sensitive value.
- SC-DCA distinguisher:

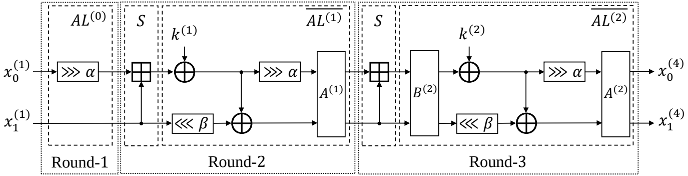
$$\delta_k^{\mathtt{SC-DCA}} = \mathtt{arg\ max} \sum_{1 \leq i \leq n} \mathtt{max}_{1 \leq j \leq t} \left|\mathtt{Cor}((\varphi_k(x))_i, \boldsymbol{v}_j)\right|$$

SC-DCA on (White-Box) SPECK

- SC-DCA on SPECK32:
  - can successfully distinguish the correct key with the maximum summed correlation 16.
  - Time complexity 2^{38} with 4096 traces.
- SC-DCA on white-box SPECK32:
  - extracts an incorrect key for both SE-SPECK32 and IF-SPECK32.
  - does not consider the encoding phases which obfuscate the sensitive values against DCA.

Encoded Structure of SE-SPECK

The first three encryption rounds of SE-SPECK without external encodings:

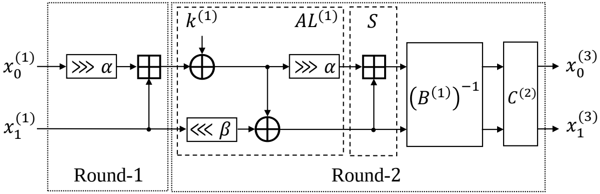


Encoded Structure of IF-SPECK

- Without external encoding, the first encoded round function is defined as

$$\overline{E^{(1)}} = C^{(2)} \circ E^{(1)} \circ A^{(1)} = C^{(2)} \circ (B^{(1)})^{-1} \circ E^{(1)}.$$

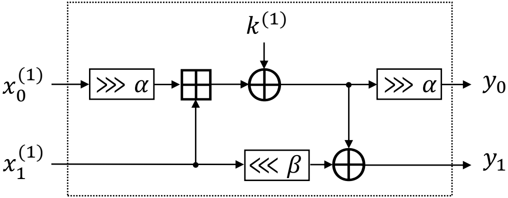
- The encoded structure of the first two rounds:



- If $B^{(1)}$ is affine, $\overline{E^{(1)}}$ is encoded by affine encoding.
- If $B^{(1)}$ is quadratic, $\overline{E^{(1)}}$ is encoded by non-linear encoding with unknown (low) degree.

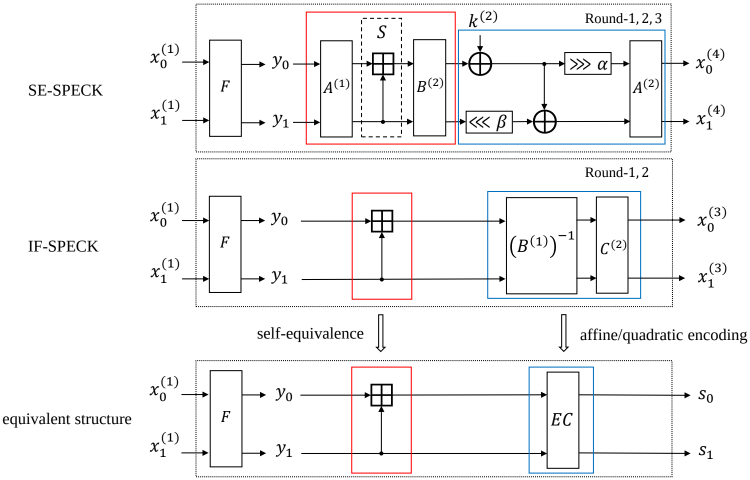
A Same Structure of SE-SPECK and IF-SEPCK

- By analyzing the round functions of SE-SPECK and IF-SPECK without external encodings, we can obtain a function F which has the same structure as the first two rounds in both two white-box implementations.



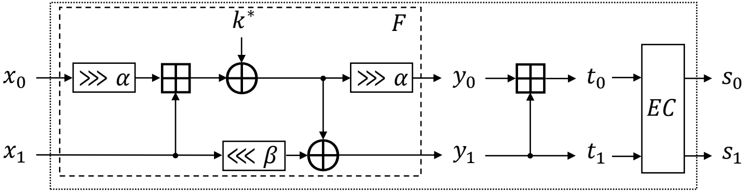
An Encoded Structure of SE-SPECK and IF-SEPCK

- Combining the first three rounds of SE-SPECK and the first two rounds of IF-SPECK:



An Encoded Structure of SE-SPECK and IF-SEPCK

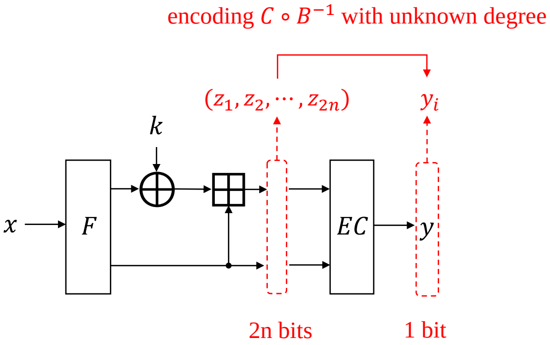
- The target function of SE-SPECK and IF-SEPCK:



- The collected computation traces consist of (s_0, s_1).
- The sensitive values (t_0, t_1) are protected by a linear/non-linear encoding EC.
- Its construction is irrelevant to the implicit function and the input encoding.

CA on SE-SPECK and IF-SPECK

- With a key guess k and an input x, CA computes a sensitive vector $(z_1, z_2, \cdots, z_{2n})$.
- CA intends to computes the correlation between z_i and y_i.
- Because of encoding EC, CA needs to enumerate the combination of $(z_1, z_2, \cdots, z_{2n})$.



Challenges of CA against White-Box ARX

- Large spaces of inputs and key candidates: CA needs to compute the outputs of the modular addition based on the inputs and key guesses.
- Large encoding: CA needs to enumerate the linear combination of the sensitive values to recover the affine encoding. Moreover, it is hard to defeat the quadratic encoding.

Outline

An Overview of CP-CA

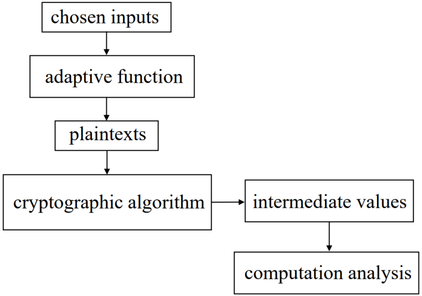
A *Chosen-Plaintext Computation Analysis* (CP-CA) attack consists of the following three phases:

- Adaptive chosen plaintexts: constructs an adaptive function with a key guess to compute the plaintexts with chosen inputs (principal: a small subset of the full space).
- Correlation computation: invokes the algorithm with the obtained plaintexts and applies the existing computation analysis methods, such as DCA and ADCA to analyze the computation traces.
- Iterative attack: repeats for other key candidates. Distinguishing the correct key based on the ranking method of the corresponding computation analysis.

Adaptive Function

An adaptive function:

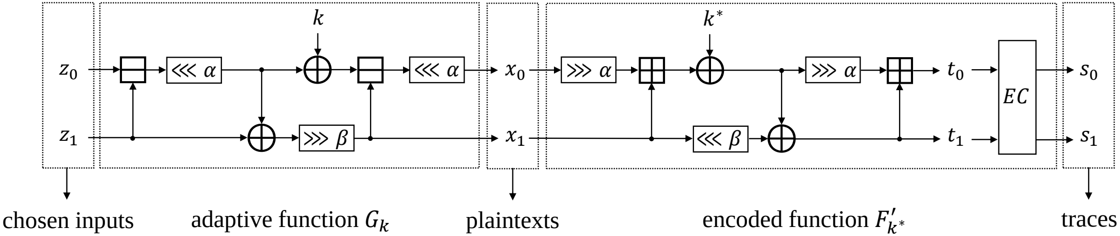
- is applied to choose the plaintexts of the cryptographic algorithm.
- is constructed by some reverse steps of the cryptographic algorithm.
- needs to be instantiated by a key guess.



CP-CA

Adaptive Function

- The adaptive function G_k calculates the plaintexts (x_0, x_1) by the inverse of (t_0, t_1).



- If key guess k equals to the correct key k^*, $(t_0, t_1) = (z_0, z_1)$.
- Otherwise, (t_0, t_1) are almost random values.

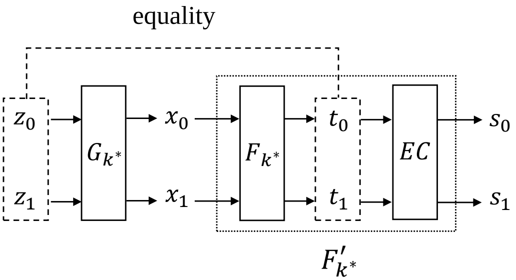
(In)Correct Key Guess

If key guess is correct:

- The collected values (s_0, s_1) are equal to the outputs of an affine/non-linear function $EC(z_0, z_1)$.
- If EC is affine, (s_0, s_1) are linear combinations of (z_0, z_1).
- If EC is non-linear, (s_0, s_1) can be represented by a degree-d ANF of (z_0, z_1).

If key guess is incorrect:

- (s_0, s_1) are correlated to random (t_0, t_1).



Affine Self-Equivalence

Theorem

Let $z \in \{0,1\}^{n_a}$ denote an n_a-bit vector, c be an n-bit constant, and C_0 represent an $(n_b\ (= n - n_a))$-bit zero vector. Given an $2n$-bit affine function $AE : \mathbb{F}_2^{2n} \mapsto \mathbb{F}_2^{2n}$, the resulting vector $AE(C_0 \parallel z, c)$ can also be computed as $L' \cdot z \oplus l$, where L' is a $2n \times n_a$ matrix and $l \in \{0,1\}^{2n}$.

- If EC is affine, (s_0, s_1) are linear combinations of z.
- For SPECK32,

$$(z_0, z_1) = (0000000000000 \parallel z, \texttt{00ff00ff00ff00ff})$$

- $(s_0, s_1) = EC \cdot (z_0, z_1) = AE \cdot (z)$ for some unknown affine function AE.

Quadratic Self-Equivalence

- If EC is quadratic, (s_0, s_1) are still linear combinations of z.
- Because of these fixed input bits, some variables in the monomials of the ANF representations of EC are constants with degree 0.
- For an instance of the degree-2 quadratic encoding case, the probability p that a monomial in the ANF has degree 2 is

$$p = \binom{n_a}{2} \Big/ \binom{2n}{2} = \frac{n_a(n_a - 1)}{2n(2n - 1)}.$$

Block size	32	48	64	96	128
p	5.65%	2.48%	1.39%	0.61%	0.34%

Outline

CP-DCA

- CP-DCA calculates the correlation between a subset of linear combinations of the chosen inputs and the samples of traces.
- The key guess with the maximum number of the highest correlation is the most likely correct.

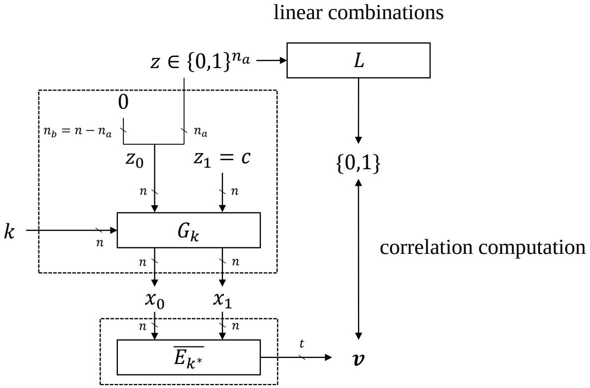
Corollary

Let y_i $(1 \leq i \leq 2n)$ denote the output coordinate of $AE(C_0 \parallel z, c)$. There exist $2n$ linear combinations L of z satisfying $|\texttt{Cor}(L \cdot z, y_i)| = 1$.

CP-DCA Distinguisher

- The CP-DCA distinguisher $\delta_k^{\mathtt{CP-DCA}}$ is defined as follows.

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



Time Complexity of CP-DCA

- Traces collection: $\mathcal{O}(|\mathcal{K}| \cdot N)$ with $|\mathcal{K}| \leftarrow$ key space, $N \leftarrow$ input space.
- Correlation computation: $\mathcal{O}(|\mathcal{K}| \cdot 2^{n_a} \cdot t \cdot N)$ with 2^{n_a} linear combinations, $t \leftarrow$ the number of trace samples.
- Searching for the highest correlation: $\mathcal{O}(|\mathcal{K}| \cdot 2^{n_a})$.
- Overall time complexity: $\mathcal{O}(|\mathcal{K}| \cdot 2^{n_a} \cdot t \cdot N)$.

CP-ADCA

- The degree-1 CP-ADCA constructs a linear system that consists of the coordinates of $x \in \mathcal{X}$ and each sample of $\boldsymbol{v}$.
- The key guess with the maximum number of solvable linear systems is the most likely correct one.

Corollary

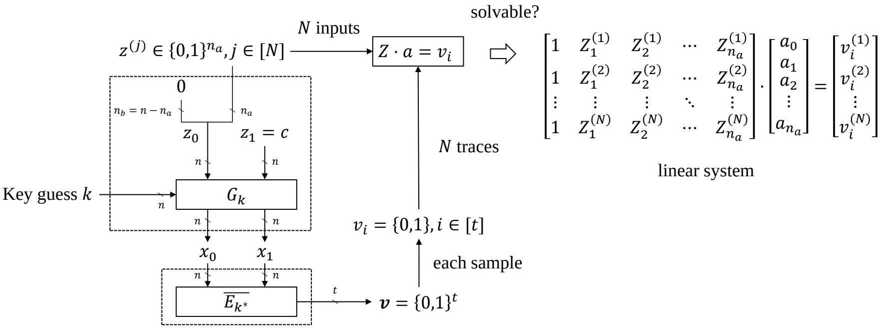
Let y_i $(1 \leq i \leq 2n)$ denote the output coordinate of $AE(C_0 \parallel z, c)$. Let Z_j $(1 \leq j \leq n_a)$ denote the bits of z. There exist $2n$ vectors $a = (a_0, a_1, \cdots, a_{n_a})$ satisfying

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

CP-ADCA Distinguisher

- The CP-ADCA distinguisher $\delta_k^{\mathtt{CP-ADCA}}$ is defined as follows.

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



Time Complexity of ADCA

- Traces collection: $\mathcal{O}(|\mathcal{K}| \cdot N)$ with $|\mathcal{K}| \leftarrow$ key space, $N \leftarrow$ input space.
- Computation of linear systems: $\mathcal{O}(|\mathcal{K}| \cdot t \cdot N \cdot (n_a + 1))$ with the steps for calculating $r(Z \mid v_i)$ are $N \cdot (n_a + 1)$, $t \leftarrow$ the number of trace samples.
- Searching for the maximum number of solvable linear systems: $\mathcal{O}(|\mathcal{K}|)$.
- Overall time complexity: $\mathcal{O}(|\mathcal{K}| \cdot t \cdot N \cdot (n_a + 1))$.

Parameters of CP-DCA and CP-ADCA

- CP-CA can be instantiated with different parameters, such as the chosen input space n_a, the constant input c, the number of traces N, the number of trace samples t.

Block size	n_b	n_a	c	N	t	Time complexity			
						CP-DCA	CP-ADCA	DCA	AC
32	8	8	`00ff`	256	32	2^{37}	$2^{31.32}$	2^{69}	$2^{30} \sim 2^{45}$
48	16		`ff00ff`		48	$2^{45.58}$	$2^{39.90}$	$2^{101.58}$	$2^{33} \sim 2^{50}$
64	24		`00ff00ff`		64	2^{54}	$2^{48.32}$	2^{134}	$2^{36} \sim 2^{54}$
96	40		`00ff00ff00ff`		96	$2^{70.58}$	$2^{64.90}$	$2^{198.58}$	$2^{39} \sim 2^{59}$
128	56		`00ff00ff00ff00ff`		128	2^{87}	$2^{81.32}$	2^{263}	$2^{42} \sim 2^{63}$

Simulations

- Performing the simulations of CP-DCA and CP-ADCA against the 32-bit encoded structure with affine output encodings.
- CP-DCA and CP-ADCA can successfully recover the secret key for the 32-bit block size.

Attack	Block size	Key guess		Count of recovered	
		Range	Count	Key	Encoding
CP-DCA	32	0000 – `ffff`	2^{16}	1	32
CP-ADCA					

SE-SPECK and IF-SPECK implementations

- Implement SE-SPECK and IF-SPECK with block sizes 32 and 48 thorough the open-source scripts.
- Intel Core i7-11800H processor @2.30GHz and 40GB RAM.

Cipher	Encoding	Degree	Source Code size (MB)	Binary size (MB)	RAM (MB)	Execution time (ms)
SE32/K64	affine	-	0.08	0.04	1.11	0.06
SE48/K96	affine	-	0.18	0.07	1.17	0.14
IF32/K64	affine	2	0.16	0.15	3.08	2.43
	quadratic	2	0.16	0.15	3.15	2.46
		3	1.85	1.82	5.30	11.63
		4	17.45	17.41	24.43	83.33

Practical attacks of CP-DCA and CP-ADCA

- CP-DCA and CP-ADCA can successfully distinguish the secret key over the full key space.

<table>
<tr><th rowspan="3">Cipher</th><th rowspan="3">Encoding</th><th rowspan="3">Degree</th><th colspan="2">CP-DCA</th><th colspan="2">CP-ADCA</th></tr>
<tr><th colspan="4">Count of recovered</th></tr>
<tr><th>Key</th><th>Encoding</th><th>Key</th><th>Encoding</th></tr>
<tr><td>SE32/K64</td><td>affine</td><td>-</td><td rowspan="6">1</td><td>18</td><td rowspan="6">1</td><td>19</td></tr>
<tr><td>SE48/K96</td><td>affine</td><td>-</td><td>25</td><td>36</td></tr>
<tr><td rowspan="4">IF32/K64</td><td>affine</td><td>2</td><td rowspan="4">32</td><td rowspan="4">32</td></tr>
<tr><td rowspan="3">quadratic</td><td>2</td></tr>
<tr><td>3</td></tr>
<tr><td>4</td></tr>
</table>

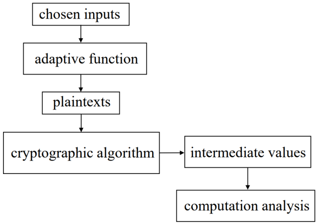
Compare with Chosen-Plaintext SCA

CP-CA:

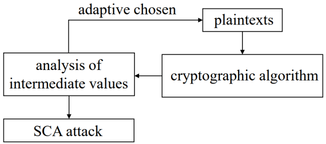
- constructs an adaptive function $\rightarrow$ computes the target plaintexts.

Adaptive Side-Channel Analysis (ASCA):

- analyzes the side-channel information $\rightarrow$ choose the target plaintexts.



CP-CA



ASCA

Possible Countermeasure of CP-CA

- The quadratic non-linear encoding can be bypassed by the adaptive function of the first-degree CP-CA.
- The possible countermeasure is to apply higher-degree non-linear encoding.

The open problems:

- The method to generate the higher-degree self-equivalences of modular addition?
- The resistance of higher-degree non-linear encoding against higher-degree CP-CA?

Possible Improvement of CP-CA

- The most optimal choice for the parameters of ACP-DCA.
  - The vector space of linear combinations,
  - the number of required traces,
  - and the constant inputs.

- Small key space. It costs a higher time complexity in the large block size cases.

- A specific analysis dedicated to the sparse affine self-equivalences.

Outline

Conclusion

- The large spaces of inputs, key candidates, and encodings of ARX-based white-box ciphers can prevent a practical DCA attack.
- CP-CA attacks exploit the chosen plaintexts phase to reduce the large affine encoding into small linear one.
- The adaptive function can bypass the quadratic self-equivalence of IF-SPECK.
- SE-SPECK and IF-SPECK are vulnerable to CP-CA attacks.

Thanks for your attention!