Side-Channel Evaluation on Protected Implementations of Several NIST LWC Finalists

Dawu Gu, Pei Cao, Yuhang Ji, Xiangjun Lu, Shipei Qu, Tengfei Wang,
Chi Zhang, Hongyi Zhang, Xiaolin Zhang (sorted alphabetically by last name)
Cryptology and Computer Security Laboratory (LoCCS)

School of Electronic Information and Electrical Engineering
Shanghai Jiao Tong University
Shanghai, China

August 12, 2022

On the Side Channel Leakage Assessment of First-Order Masked Romulus

Xiaolin Zhang[1], Tengfei Wang[1] and Pei Cao[1]

School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China

1 Introduction

1.1 Background

Romulus[CIK+22] is a family of tweakable block cipher-based authenticated encryption (AE) schemes. It has been selected as one of the finalists in the NIST lightweight cryptography standardization process. It consists of a nonce-based AE Romulus-N (the main variant), a nonce misuse-resistant AE Romulus-M, a leakage-resilient AE Romulus-T, and a hash function Romulus-H. Romulus achieves beyond-birthday-bound security, and it is also computationally efficient in both software and hardware implementations. However, its performance on power side-channel analysis is yet to be explored.

Power side-channel analysis enables attackers to collect the power consumption of a cryptographic hardware device, which allows them to infer the secrets inside, e.g. private keys. More precisely, simple power analysis (SPA) refers to interpreting raw power traces visually to deduce the patterns of cryptographic operations. Differential power analysis (DPA)[KJJ99] is an advanced technique based on statistical analysis, which helps attackers to reveal the key through intermediate values of the cryptographic computations. Over the decade, deep learning (DL) has been developed as a powerful tool for side-channel attacks.

In this report, we will perform a side-channel leakage assessment against Romulus with first-order boolean masking in software and hardware implementations. The collected power traces are going through several tests such as Welch's t-test and correlation power analysis (CPA) to demonstrate the actual performance of the side-channel resilience of Romulus.

1.2 Our Work and Results Overview

Our work in this report and the results of the side-channel leakage assessment on first-order masked Romulus can be summarized as follows.

- We collected four trace sets from the given software and hardware implementations of Romulus-N on an MCU and a side-channel attack evaluation board.
- We performed Welch's t-test, χ^2-test and DL-LA to evaluate the power leakage of Romulus. We tried to recover the private keys of Romulus-N by CPA and template attack.
- Welch's t-test, χ^2-test and DL-LA applied on the power traces from the given implementations both show the potential power leakage from the input nonce.

- CPA and template attack cannot recover the private key bytes under the given implementations.

The overall experimental results show that there exists potential power leakage related to the input nonce of Romulus, but the private key and its corresponding intermediate values do not exhibit noticeable leakage for side-channel analysis.

The rest of this report is organized as follows. Section 2 introduces our assessment strategy on Romulus. Section 3 gives the detailed experimental settings. Section 4 presents the basic information about the collected power traces and the main test results are shown in Section 5.

2 Assessment Strategy

Our assessment strategy on the given Romulus implementations can be boiled down into the following three phases:

Phase 1: Specify the analysis targets. We choose Romulus-N as the evaluated algorithm for Romulus since it is the basis of Romulus-M and Romulus-T and shares the same building block (`Skinny-128-384+`) with Romulus-H. Therefore, its performance on the side-channel analysis can be viewed as the general results of the Romulus family.

Next, we chose the output of `AddRoundTweakey` in the first round of `Skinny-128-384+` as the intermediate value. The state matrix in the first round is XORed with the tweak key $(TK1, TK2, TK3)$ and we have $TK_2 = N$ and $TK_3 = K$, where N, K denote the nonce and the encryption key, respectively. Then the operation `AddRoundTweakey` now can be written as (1).

$$\begin{aligned}\mathsf{state}^{'} &= \mathsf{state} \oplus TK1 \oplus TK2 \oplus TK3 \\ &= \mathsf{state} \oplus TK1 \oplus N \oplus K\end{aligned} \tag{1}$$

The reason that we do not chose the output of SBox in `Skinny-128-384+` as intermediate values is that $\texttt{state}^{'}$ needs to go through `ShiftRows` and `MixColumns` before it is fed into the Sbox as shown in Figure 1. Then each byte of the Sbox output can relate to multiple input bytes, which undermines the efficacy of CPA and other tests.

Therefore, though the non-linearity of Sbox can ease the side-channel analysis, we select the output of `AddRoundTweakey` in the first round, the result of a linear operation XOR, as the analysis targets due to `Skinny-128-384+`'s special construction.

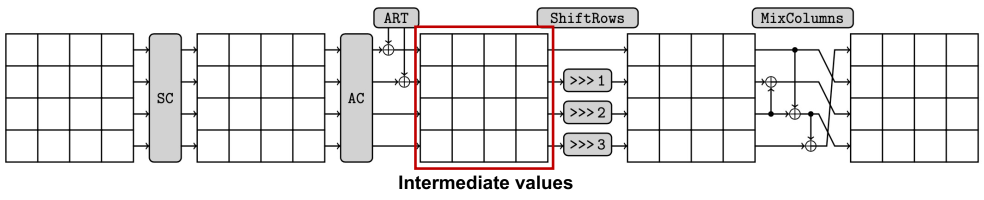


Figure 1: One encryption round of `Skinny-128-384+` (from Romulus documentation)

Phase 2: Detect side-channel leakage. We then follow the paradigms of TVLA (Test Vector Leakage Assessment) to determine whether there is noticeable power leakage in the collected raw traces. Specifically, the main techniques used here are Welch's t-test and χ^2 test. They can roughly locate where in the traces the power leakage occurred and thus indicate which cryptographic operations in Romulus caused that leakage.

Phase 3: Reveal the secret key. Note that if there is power leakage detected in **Phase 2**, we can apply CPA here to reveal the private key byte-by-byte. We will also use the template attack (TA), a traditional profiled side-channel analysis approach, and try to explore more information about the key.

To apply the above assessment strategy, we need to collect enough power traces of protected implementations of Romulus-N on specific hardware.

3 Experimental Setup

3.1 Overall Procedure

The procedure of out power trace collection experiments is presented in Figure 2.

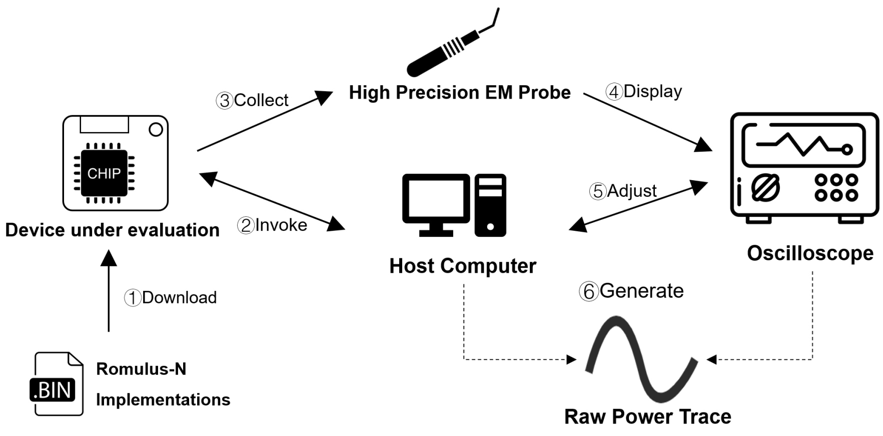


Figure 2: Overall procedure of power trace collection

We first need to download the firmware containing the C/ASM implementation of Romulus-N into the device's flash memory. Then we connect the device to the host computer through a USB serial port so that we can execute the cipher and record its intput and output. Meanwhile, we use a high-precision electromagnetic probe to capture the electromagnetic power emitted from the device chip. The captured power is then transmitted to the oscilloscope to generate and display the waveform of electronic signals.

With the help of the oscilloscope, we can acquire enough raw power traces of protected Romulus-N in the host computer for later assessment.

3.2 Experimental Setting

We can follow the above procedure to build an automatic power trace collection platform for Romulus. However, several practical problems need to be considered here.

3.2.1 Trigger location

Apart from the equipment mentioned in Figure 2, another probe attached to the oscilloscope can receive trigger signals to help us locate the timing when Romulus-N is executed. Thus, we need to modify the original Romulus-N implementations so that they can control the corresponding pins of the device to send trigger signals to the oscilloscope.

Software implementation. The C/ASM codes to control the pin and send the trigger signals are inserted into the the following two locations.

- Prior and after the call to crypto_aead_encrypt_shared;

- Prior and after the call to the first quadruple_round instruction of Skinny encryption function in skinny128_core.s.

These code snippets to manipulate pins of the target device are shown in Figure 3.

```
// call trigger
HAL_GPIO_TogglePin(Trigger_GPIO_Port,
Trigger_Pin);
// encryption
crypto_aead_encrypt_shared(…);
// call trigger
HAL_GPIO_TogglePin(Trigger_GPIO_Port,
Trigger_Pin);
```

(a) Codes around Romulus encryption function

```
push     {r0-r12, r14} // save contexts
mov.w    R1, #0x1000
mov.w    R0, #0x48000000
BL       HAL_GPIO_TogglePin // call trigger
pop     {r0-r12, r14}
quadruple_round
push     {r0-r12, r14}
mov.w    R1, #0x1000
mov.w    R0, #0x48000000
BL       HAL_GPIO_TogglePin  // call trigger
pop     {r0-r12, r14}
```

(b) Codes around Skinny encryption round instruction

Figure 3: Code snippets to set triggers in the software implementation

Note that the function that we insert trigger codes in Figure 3(b) into is `skinny128_384_plus_enc`, a copy from the original `skinny128_384_plus`. Then we let `romulusn_process_msg` call `skinny128_384_plus_enc` instead of `skinny128_384_plus`. In other words, plaintext encryption would call different functions from the associated data. Thus, we can isolate the target for side-channel analysis.

Hardware implementation. The verilog code snippets in Figure 4(a) shows the trigger signals we assigned in the hardware implementation. Figure 4(b) gives the settings of Romulus-N verilog module instantiations.

```
// trigger signal
assign gpio_startn = ~drdy;
assign gpio_endn = ~pdi_valid;
assign gpio_exec = ~sdi_valid;

// custom settings
assign pdi_valid = (state & (rom_addr == 0
| (rom_addr > 6 & rom_addr < 23)))? 1 : 0;
assign sdi_valid = (state & (rom_addr > 0
& rom_addr <= 6))? 1 : 0;
```

(a) assigned trigger signals

```
inst_rom inst_rom_0(
.addr(rom_addr), .din(blk_din),
.dout(rom_dout));
LWC_SCA LWC_SCA_0 (
// Outputs
.pdi_ready(pdi_ready), .sdi_ready(sdi
_ready),
.rdi_ready(rdi_ready), ……
// Inputs
.pdi_valid(pdi_valid), .sdi_valid(sdi
_valid), ……);
```

(b) verilog module instantiation

Figure 4: Code snippets to set triggers in the hardware implementation

Through the indication of `gpio_exec`, we can adjust the oscilloscope to sample raw power trace with any range.

3.2.2 Input and output of Romulus-N

During the experiments of power trace collection, the input of Romulus-N encryption consists of three parts: a 16-byte nonce, 16-byte associated data and 16-byte plaintext. The output consists of 16-byte ciphertext and a 16-byte authenticated tag. The 16-byte encryption key is fixed throughout the collection. The specific information about the fixed input is shown in Table 1.

According to the analysis in Section 2, changing either the input nonce or plaintext will change the intermediate values. Here we choose to alter the nonce in each encryption. Then the intermediate values will change under the same key, thereby generating different but related power consumption patterns. This allows us to perform CPA and other tests.

Table 1: Input details of Romulus-N

Platform	Fixed Input	Value
Software implementation	Private key	`000102030405060708090A0B0C0D0E0F`
	Plaintext	`000102030405060708090A0B0C0D0E0F`
	Associated data	`000102030405060708090A0B0C0D0E0F`
Hardware implementation	Private key	`4535819F13209B89C4C604385A87F47E`
	Plaintext	`BFFE6A4BD1DFE787E9D9E8AC5AEFFC74`
	Associated data	`2B71FF688E9188E145FB95AB12BF19C9`

3.2.3 Experimental environments

The details of devices and analyzing suites used for Romulus are presented in Table 2.

Table 2: Details of experimental environments

Type	Items	Details
Hardware platform	Target MCU	STM32F303RCT6
	Target evaluation board	Saseabo-giii (with Xilinx Kintex-7 FPGA)
Measuring tools	High Precision EM probe	Langer RF-U 5-2
	Oscilloscope	Pico 3203D, LeCroy 610Zi
Sampling parameters	Baud rate (USB Serial Port)	115200 bps
	Sampling rate	125 MHz, 500 MHz
Random source	standard C library	`rand()`, `srand()` in `stdlib.h`

We assign `GPIO_12` of STM32F303RCT6 (`CN9` of Saseabo-gii) as the pin sending the trigger signals. The given software and hardware implementations of Romulus-N will be tested on STM32F303RCT6 and Saseabo-giii, respectively.

4 Description of Collected Raw Traces

We collected four sets of power traces, (S-1), (S-2), (H-1) and (H-2). (S-1) and (S-2) are sampled from the given software Romulus-N implementations under different trigger settings introduced in Section 3.2.1, and (H-1), (H-2) are from the hardware implementation. Their basic information is presented in Table 3.

Table 3: Basic information of the collected power traces of Romulus-N

Source	Software Implementation		Hardware Implementation	
Trace set ID	S-1	S-2	H-1	H-2
Skinny rounds contained	40	4	40	3
No. of traces	100000	100000	100000	1000000
No. of points per trace	20000	1800	100000	5000
Precision	$-2^{15} \sim 2^{15}$	$-2^{15} \sim 2^{15}$	$-2^{7} \sim 2^{7}$	$-2^{7} \sim 2^{7}$
Sampling time	3h	2h	1h	10h

The sample graphs of trace set (S-1) and (H-1) are presented in Figure 5. As seen from Figure 5(b), we can easily distinguish 40 rounds in Skinny encryption from (H-1).

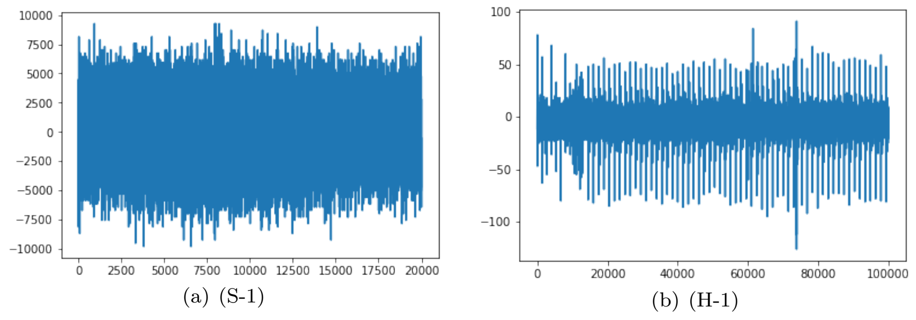


Figure 5: Sample graph of trace set (S-1) and (H-1)

Though it is hard to acquire useful information from (S-1) visually, (S-2) could show the first four rounds of encryption as shown in Figure 6(a).

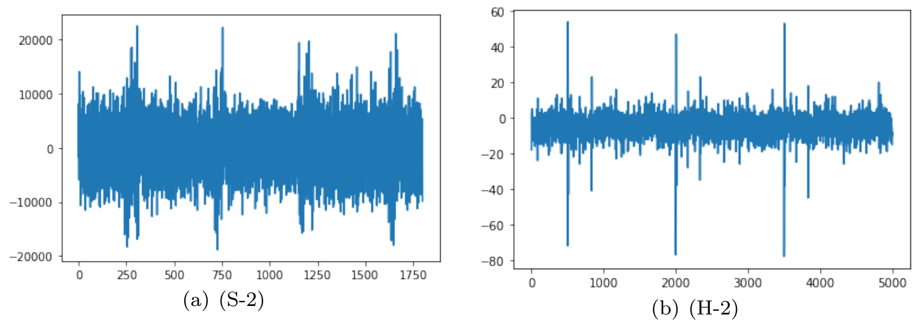


Figure 6: Sample graph of trace set (S-2) and (H-2)

We can see that (S-2) and (H-2) both show the clear power consumption tracks of Romulus-N. Then we can perform different tests mentioned in Section 2 on them to evaluate the power leakage of the given implementations.

5 Main Results

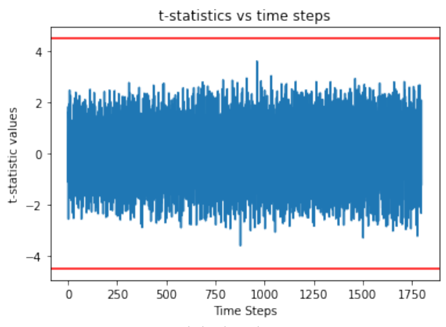
Since (S-2) and (H-2) has better sampling quality and smaller dimensions, all the tests below will be performed on them.

5.1 Welch's t-test

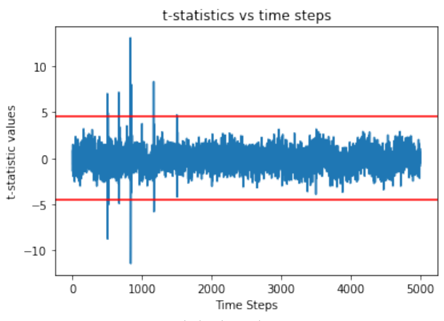
Welch's t-test is a statistical hypothesis test used to compare the means of two groups, especially when the two groups have unequal sample sizes and variances. In terms of side-channel analysis, we can divide the power traces into two groups according to the difference in intermediate values.

More precisely, when the private key is fixed, we can divide the power traces of Romulus-N by the following two cases.

- **Case(A)**: The last bit of the first byte of the input nonce is 0 or 1.
- **Case(B)**: The last bit of the first byte of the intermediate value is 0 or 1.

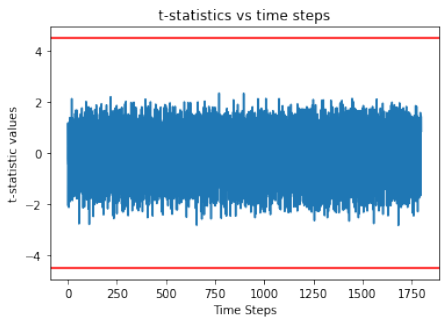


(a) (S-2)

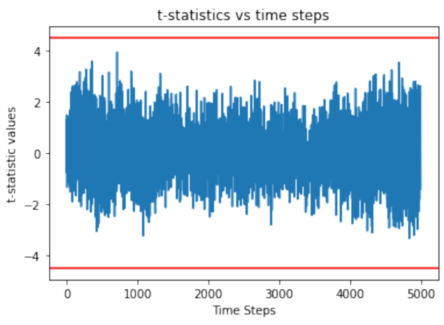


(b) (H-2)

Figure 7: Welch's t-test results of (S-2) and (H-2) (divided by Case (A))



(a) (S-2)



(b) (H-2)

Figure 8: Welch's t-test results of (S-2) and (H-2) (divided by Case (B))

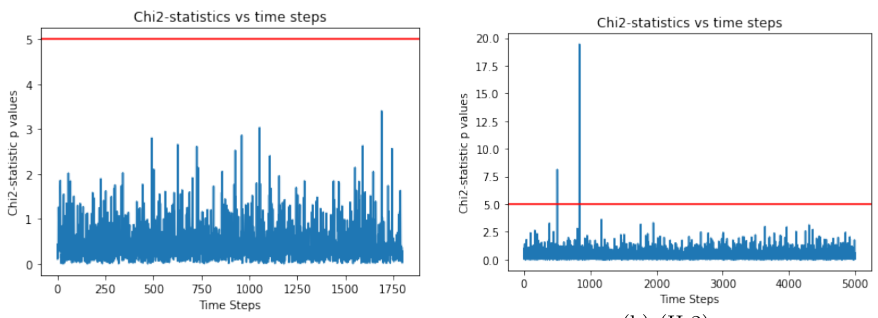
The test results are shown in Figure 7 and 8. We can see that the results of (S-2) fail to achieve the confidence level of Welch's t-test, but there is significant power leakage detected from (H-2) in Figure 7(b) when the traces are divided by nonce. Moreover, the overall range causing leakage matches the position where TK is involved in the first two encryption rounds. However, such leakage is missing in Figure 8(b).

5.2 χ^2-test

χ^2-test is a statistical hypothesis test to determine whether there is a significant difference between the expected and observed frequencies. It can also test the null hypothesis of independence of a pair of random variables. Therefore, like t-test, we need to divide these power traces by the following two cases and observe their statistical differences.

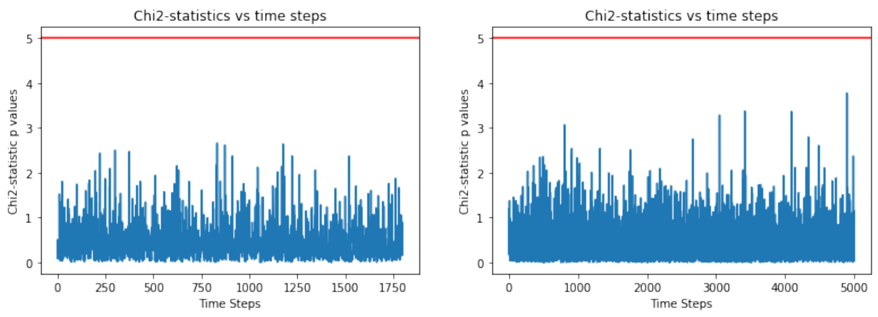
- **Case (A)**: The last bit of the first byte of the input nonce is 0 or 1.
- **Case (B)**: The last bit of the first byte of the intermediate value is 0 or 1.

Figure 9(b) shows that χ^2-test can also detect significant power leakage from (H-2), and the time steps causing leakage are approximately the same as Figure 7(b). It suggests that there exist potential power side-channel issues for the given hardware implementation of Romulus-N. However, when the traces are divided by the intermediate values, $\chi^2 - test$ cannot find statistically significant differences of two trace groups.



(a) (S-2) (b) (H-2)

Figure 9: χ^2-test results of (S-2) and (H-2) (divided by Case (A))

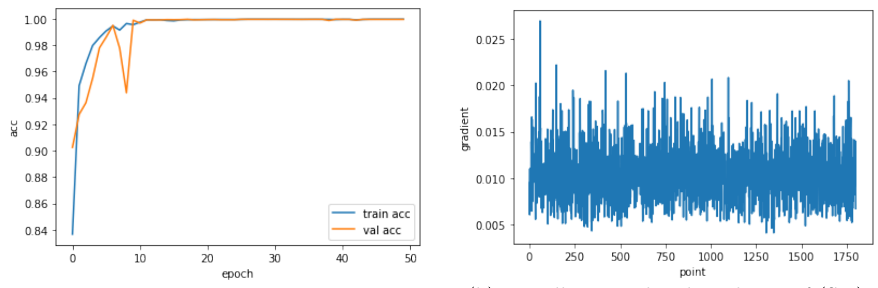


(a) (S-2) (b) (H-2)

Figure 10: χ^2-test results of (S-2) and (H-2) (divided by Case (B))

5.3 DL-LA

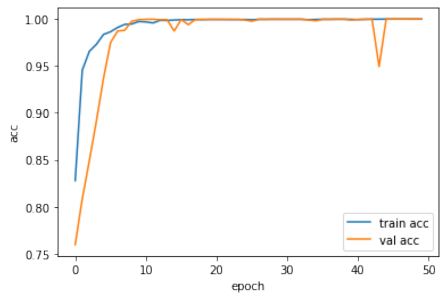
DL-LA (Deep Learning Leakage Assessment) [MWM21] is based on the concept of supervised learning and uses neural networks to build a binary classifier on power side-channel measurements. The power traces are first separated into the training and validation set. Then we need to divide them as in Section 5.1 and 5.2 for labeling. The "labels" of each power trace would stand for which group the trace belongs to.



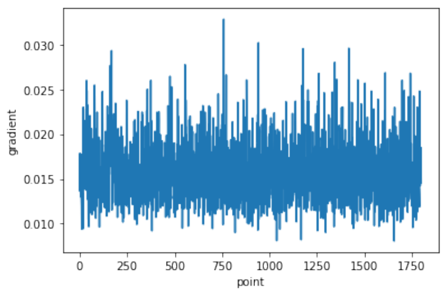
(a) accuracy of the CNN model (b) overall accumulated gradients of (S-2)

Figure 11: Assessment results of DL-LA on (S-2) (divided by Case (A))

After the classifier model is built with the labeled training set, it can be applied to the validation set to predict which group each trace belongs to. DL-LA also introduces sensitivity analysis to calculate the accumulated gradients, thereby showing where in each trace cause the bias of different groups. Here we chose the traditional CNN (Convolutional Neural Network) model and the assessment results of (S-2) are shown in Figure 11 and 12.



(a) accuracy of the CNN model



(b) overall accumulated gradients of (S-2)

Figure 12: Assessment results of DL-LA on (S-2) (divided by Case (B))

We can see that the CNN classifier can achieve over 98% accuracy in both cases, which shows that the divided two groups indeed have a distinctive statistical difference. Figure 11(b) and 12(b) show the distribution of overall gradients during the training of the CNN model. They can only roughly indicate the positions that cause the leakage. Thus, like Welch's t-test and χ^2-test, DL-LA can also detect power leakage from the given Romulus-N implementations.

5.4 CPA

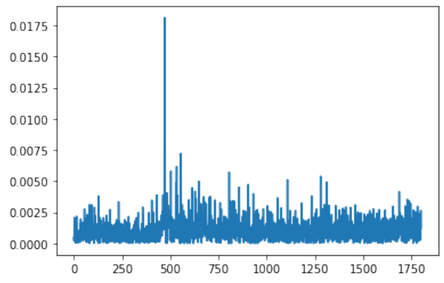
CPA is an efficient side-channel analysis technique to reveal the private keys using power leakage of a cryptographic device. It usually involves modelling the simulated power consumption under a fixed key. For each subkey byte, it computes all 2^8 possible intermediate values and then uses the hamming weight model to simulate the corresponding power consumption. The correct guess will exhibit the greatest level of correlation between the simulation and real power trace, which indicates the correct subkey byte.

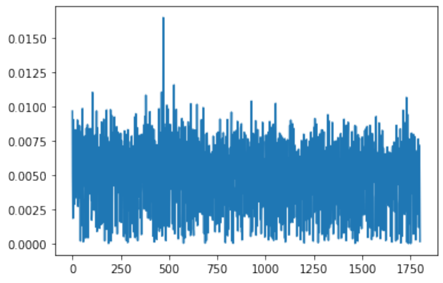
For trace set (S-2), we perform CPA on all 16 key bytes of Romulus-N. The CPA guess results for each byte is presented in Table 4.

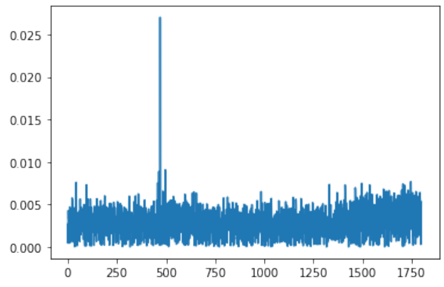
Table 4: CPA guess result of key bytes in (S-2)

Target byte index	0	1	2	3	4	5	6	7
Target byte value	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
Guess rank	2	8	7	0	98	225	89	235
Actual best guess	0xDF	0x21	0xDF	0x03	0x4B	0x69	0xBF	0x3D
Target byte index	8	9	10	11	12	13	14	15
Target byte value	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
Guess rank	191	20	90	183	90	162	88	178
Actual best guess	0x6F	0x0F	0xDF	0x81	0xF4	0xC4	0x53	0x4A

Note that first four bytes have a relatively higher rank and their correlation results are shown in Figure 13. We can also see that the key byte `0x03` can be guessed correctly using CPA. The rest of the bytes can be regarded as secure in the protected Romulus-N implementation.


(a) byte index = 0


(b) byte index = 1


(c) byte index = 2

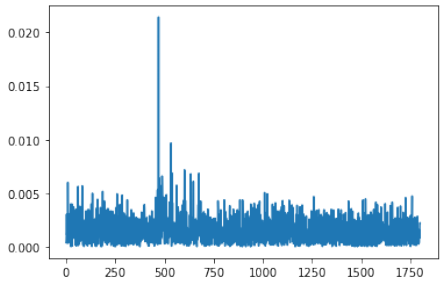

(d) byte index = 3

Figure 13: CPA correlation result of (S-2)

For (H-2), we have the following key guess results in Table 5.

Table 5: CPA guess result of key bytes in (H-2)

Target byte index	0	1	2	3	4	5	6	7
Target byte value	`0x45`	`0x35`	`0x81`	`0x9F`	`0x13`	`0x20`	`0x9B`	`0x89`
Guess rank	136	165	57	110	105	15	36	22
Actual best guess	`0x72`	`0xF9`	`0xB7`	`0x3F`	`0xC9`	`0xE0`	`0xEC`	`0x36`
Target byte index	8	9	10	11	12	13	14	15
Target byte value	`0xC4`	`0xC6`	`0x04`	`0x38`	`0x5A`	`0x87`	`0xF4`	`0x7E`
Guess rank	7	231	58	195	186	252	122	144
Actual best guess	`0xCC`	`0x8F`	`0x03`	`0xD8`	`0x1B`	`0x61`	`0x47`	`0x5C`

The correlation results of bytes `0x20`, `0x9B`, `0x89` and `0xC4` that have higher guess ranks are shown in Figure 14. According to Table 5 and Figure 14, We can see that the *fixslicing* masking scheme[BDCU17] applied in the Romulus-N implementations can prevent an attacker from recovering the correct key bytes using CPA. Thus, the power leakage presented in Section 5.1 and 5.2 could be brought by nonce rather than intermediate values.

5.5 TA

Template attack (TA) is an advanced type of side-channel attack, which needs attackers to create a power consumption template of the target device (**Profiling Phase**) and applies this template to recover the secret key efficiently (**Attacking/Predicting Phase**).

To create a template, the attacker could first perform correlation analysis on the intermediate values to acquire some points of interest (POI). Then he/she can build a

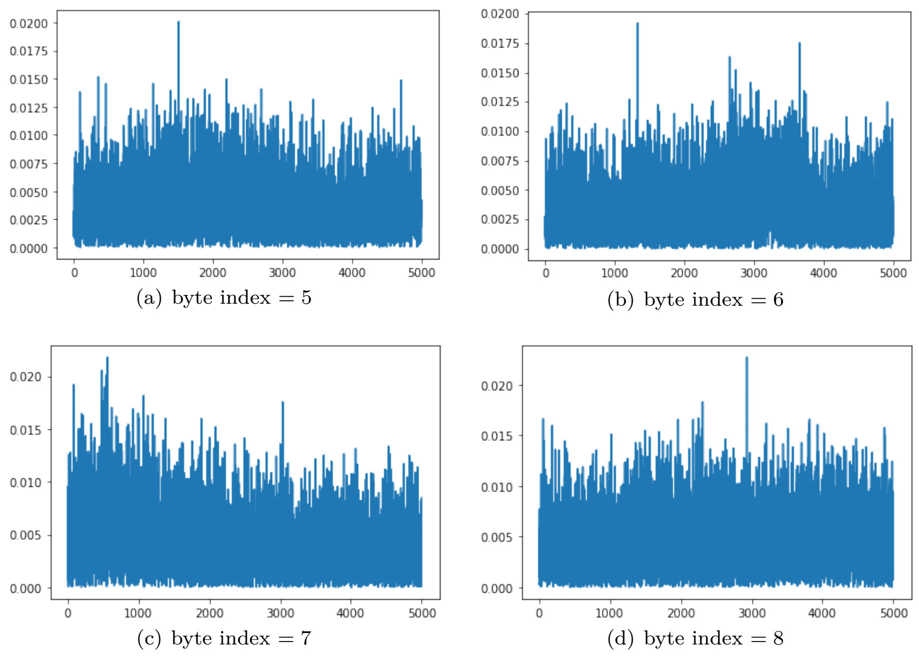


Figure 14: CPA correlation result of (H-2)

targeted template using enough power traces. Therefore, according to the results in Table 4, we can apply TA to the key byte `0x30` and `0xC4` (the highest guess rank in each trace set).

Table 6: TA prediction results of (S-2) and (H-2)

Target bytes	0x30	0xC4
Accuracy	27.33%	26.99%
Predicted value	0x02	0x51

Note that we do not consider other key bytes in TA, otherwise the built template cannot generate correct statistical features about the right intermediate values. As shown in Table 6, the accuracies of TA can only achieve 27.33% and 26.99% on the two bytes, respectively, which is close to random guessing. Therefore, the given boolean-masking implementations can protect the inside private key of Romulus-N against TA.

References

[BDCU17] Alex Biryukov, Daniel Dinu, Yann Le Corre, and Aleksei Udovenko. Optimal first-order boolean masking for embedded iot devices. In *International Conference on Smart Card Research and Advanced Applications*, pages 22–41. Springer, 2017.

[CIK$^+$22] Guo Chun, Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Romulus authenticated encryption/hash, 2022. https://romulusae.github.io/romulus.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual international cryptology conference*, pages 388–397. Springer, 1999.

[MWM21] Thorben Moos, Felix Wegener, and Amir Moradi. Dl-la: Deep learning leakage assessment: A modern roadmap for sca evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 552–598, 2021.

Side-channel Evaluation of ISAP

Yuhang Ji[1]

School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China

1 Introduction

1.1 Background

ISAP is a family of nonce-based authenticated ciphers with associated data (AEAD) designed with a focus on robustness against passive side-channel attacks. All ISAP family members are permutation-based designs that combine variants of the sponge-based ISAP mode with one of several published lightweight permutations.

Power side-channel analysis enables attackers to collect the power consumption of a cryptographic hardware device, which allows them to infer the secrets inside, e.g. private keys. More precisely, simple power analysis (SPA) refers to interpreting raw power traces visually to deduce the patterns of cryptographic operations. Dierential power analysis (DPA) is a more advanced technique based on statistical analysis, which helps attackers to reveal the original key through intermediate values of the cryptographic computations. Over the decade, deep learning (DL) has been developed as a powerful tool for side-channel attacks.

ISAP is known for its resilience against DPA attacks. Until the day of this report, there is no published work regarding ISAP's side-channel security. In this report, we perform a side-channel leakage assessment against ISAP. The collected power traces are going through several tests such as Welchs t-test and correlation power analysis (CPA) to demonstrate the actual performance of the side-channel resilience of ISAP.

1.2 Our Work and Results Overview

Our work in this report and the results of the side-channel leakage assessment on ISAP can be summarized as follows.

- We collected three trace sets from the given software and hardware implementations of ISAP-Ascon on an MCU and a side-channel attack evaluation board.
- We performed side-channel leakage assessment on ISAPRk procedure of an ISAP encryption. Welchs t-test and χ^2-test were used to evaluate the power leakage of ISAP-Ascon.
- CPA attack cannot recover the private key bytes under the given implementations.

The overall assessment reveals that power leakage mainly comes from associated data input of the ISAPRk procedure, while the actual private key and permutation does not induce any leakage.

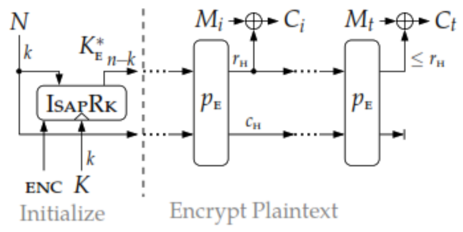


Figure 1: Authenticated Encryption of ISAP

2 Assessment Strategy

2.1 Specify the Targets

Side-channel analysis towards a cryptography algorithm normally requires three essential elements: a random variable x that's visible to attacker, a fixed secret value k the attacker wants to recover, and a non-linear operation f which takes x and k as inputs and generates sensitive values c. c is known as the sensitive intermediate values to k. Operation f on the crypto hardware generates side-channel leakages, which is know as traces. Normally the power leakage increases by the non-linearity of operation f. Side-channel analysis heavily relies on the selection of these three elements.

The ISAP algorithm provides several modes, such as authenticated encryption/decryption and MAC. All of the modes are designed to sponge-based constructions. The basic operation unit of ISAP is permutation. The states of the opted permutation are used to absorb data and squeeze data. Permutation used in ISAP can be either Ascon or Keccak. The selection of the permutation leverages on the environment the ISAP needs to be and does not pose any discrepancy in the scenarios of Side-Channel analysis. In this paper, Ascon is opted for the permutation as illustration.

Below we will analyze each component of ISAP and find out which component has a potential side-channel vulnerability. The ultimate goal is to recover the master key of Ascon based ISAP with 128 bit security parameter.

2.1.1 Authenticated Encryption

Authenticated encryption and decryption with associated data have the same structure in ISAP. In this section we focus on the encryption.

As illustrated in Figure 1, ISAP encryption takes three inputs: nonce N, master key K and message M. Nonce and master key is used to derive encryption key K_E^* using ISAP component ISAPRk. And K_E^* and nonce jointly forms the initial state of the Ascon permutation. Nonce N must vary from each encryption and is visible to the attacker. Any bit change on N will result in different K_E^* and different initial states thus affecting the ciphertexts C by avalanche effect.

What is visible to the attacker is N, M_i and C_i. Note that nonce should be assumed not to reuse in each encryption so initial states of permutation varies constantly, rendering the recovery of K_E^* impossible and meaningless. To achieve cipher text forgery, the attacker must be able to recover master key K to compute K_E^* using ISAPRk.

Due to the discarding of states variables in the end, the attacker will not be able to recover the input of final permutation p_E, thus K_E^* is invisible.

We'll dive into the ISAPRk component later.

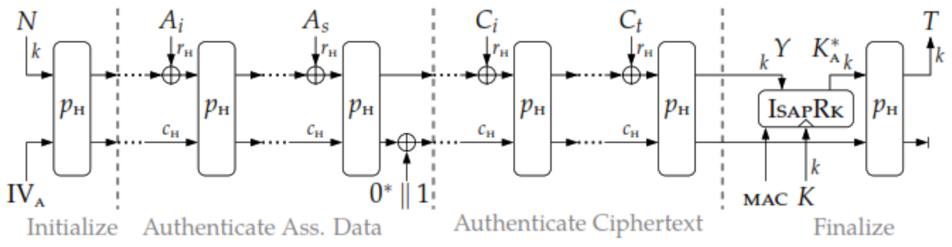


Figure 2: Message Authentication Code of ISAP

2.1.2 Message Authentication Code

As shown in Figure 2, the ISAPMAC takes nonce N, fixed public parameter IV_A, public associated data A, cipher C and master key K as inputs, generates tag T using part of the states and discards unused states in the end.

Due to the discarding of states variables in the end, the attacker will not be able to recover the input of final permutation p_H, thus K_A^* is invisible. By changing A and C, the attacker can change and compute ISAPRk parameter Y.

2.1.3 Rekey

Rekey component (ISAPRk) serves as a key derivation function using fixed master key. As we have discussed in 2.1.1 and 2.1.2, it is obvious that the attacker should target on ISAPRk to recover master key K.

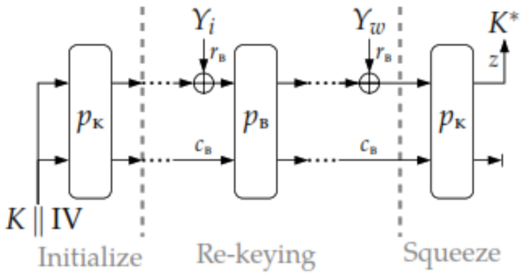


Figure 3: ISAPRk

The ISAPRk takes three inputs: the fixed master key K, the publicly visible initial vector IV and associated data Y. From both scenarios where ISAPRk is used, the attacker is always able to observe Y and IV, and always unable to observe K^*. Since the K and IV are both fixed, the state outputs by initialization stage are fixed as well. A recovery on initial states will also enable the attacker to compute K^*, thus compromising the security of ISAPEnc and ISAPMAC.

Y is split into blocks of size r_B, indicating only r_B bits of Y are absorbed by each permutation. In our scenario, $r_B = 1$ and the state bit length of 320.

Since the goal of the attacker is to recover K, she may manipulate Y and observe side-channel leakages on p_B and tries to recover the state variables.

2.2 Specify the Strategy

We will focus on the permutation p_B used in ISAPRk.

The non-linearity of Ascon is induced by the internal SBox construction. In the typical implementation, 320 bit state is split into 5 state variables of 64 bit, and data to be absorbed only influence the leading r_B bits.

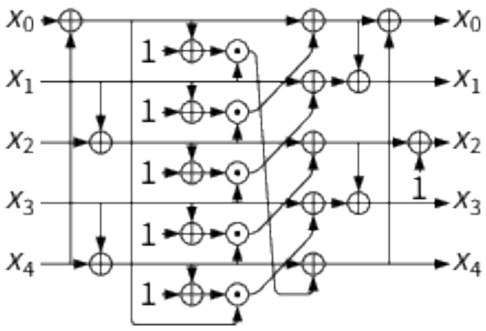


Figure 4: SBox of Ascon permutation

Before the SBox, the Ascon permutation only performs a linear constant addition to the state. And the constant is public so the attacker can calculate SBox input giving the data to be absorbed and initial state. As shown in Figure 4, state variables x_0 to x_4 to be passed to SBox are 64 bit each. All the operations are bit-wise, indicating the calculation is on bits of the same position of x_i.

For each absorption, only leading r_B bits of x_0 are randomizable. The attacker will not be able to recover trailing fixed $320 - r_B$ bits.

To recover the leading r_B bits of each x_i, the attacker can randomize leading r_B bits of Y and cause leading r_B bits of input x_0 to change and choose same bits of SBox output x_0 to x_5 as intermediate values.

Since SBox is a highly non-linear operation on input bits, any leakage will be amplified on hardware, which constitutes a target of side-channel analysis. Although only recover leading r_B bits of five state variables can be recovered, the attacker will gather sufficient information regarding of the permutation state after sufficient rounds of permutation. Having correctly recovered sufficient leading r_B bits of multiple intermediate states, the attacker can utilize a linear solver (like z3) to solve the initial state mathematically.

Our analysis strategy uses a r_B bit visible random value to recover $5 \times r_B$ fixed values by exploiting non-linear SBox which takes them as input, and the intermediate values are output leading r_B bits of five state variables. It is hard to achieve such $1 : 5$ information ratio under side-channel analysis, but it's worth trying.

3 Experiments

3.1 Setting

We rst need to download the rmware containing the C/ASM implementation of Ascon-ISAP into the devices ash memory. Then we connect the device to the host computer through a USB serial port so that we can execute the cipher and record its input and output. Meanwhile, we use a high-precision electromagnetic probe to capture the electromagnetic power emitted from the device chip. The captured power is then transmitted to the oscilloscope to generate and display the waveform of electronic signals. With the help of the oscilloscope, we can acquire enough raw power traces of ISAP in the host computer for later assessment.

The hardware platform we use to flash code into is STM32F303RCT6 with Xilinx Kintex-7 FPGA. The power consumption traces are acquired by a high precision LeCroy

610Zi osscilliscope.

The assessment we have conducted for each implementation is listed in Table 1.

Table 1: Assessment scenarios

Implementation	Assessment Strategy
Software impl. by ISAP team	CPA
Hardware impl. by IAIK	CPA
Hardware impl. by Ruhr-University Bochum	CPA, t-test, χ^2-test

Since the associated data input in both ISAPEnc and ISAPMAC are visible to attacker, so the analysis on ISAPRk precedure are expected to present the same results on them. In our experiment, we decided to run ISAPEnc and collected nonce inputs and master key for further analysis.

The acquisition triggers were placed on entering ISAPRk for each implementation.

3.2 Software Implementation by ISAP team

CPA is a side-channel analysis technique to reveal the private keys using the power leakage of a cryptographic device. We conducted CPA on the ISAP software implementation.

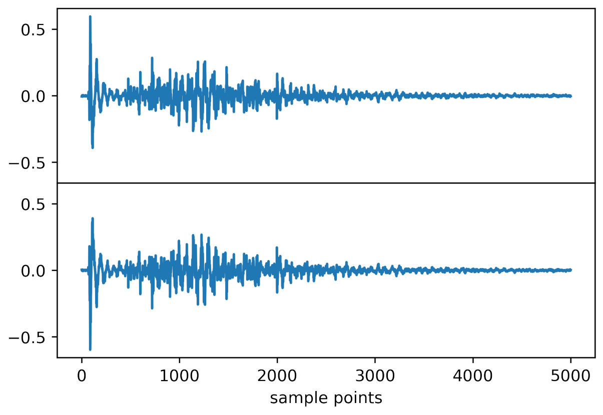


Figure 5: CPA on Software Implementation. The upper one is a random guess, and the lower one is the correct guess

The CPA result on random guess and correct guess of corresponding 5 bits of initial state does not present any notable discrepancy. So the software CPA can not effectively distinguish the correct initial state from wrong ones.

3.3 Hardware Implementation by IAIK

The results on figure 6 presents exactly the same phenomenon as software implementation. There is also no notable discrepancy between random guess and correct initial states.

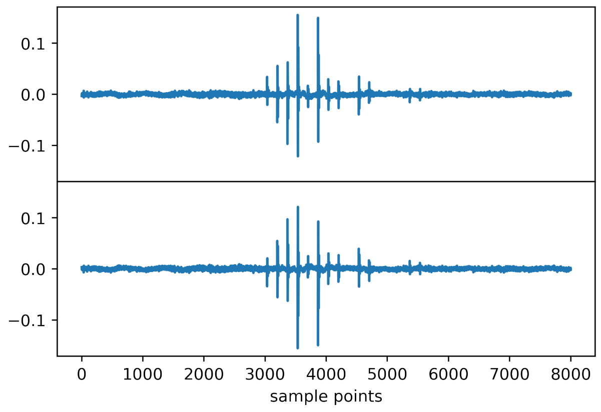


Figure 6: CPA on IAIK hardware implementation. The upper one is a random guess, and the lower one is the correct guess

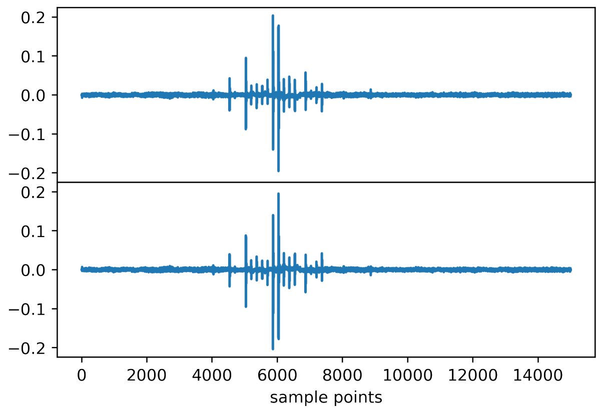


Figure 7: CPA on ISAP second order hardware implementation. The upper one is a random guess, and the lower one is the correct guess

3.4 Hardware implementation by Ruhr-University Bochum

3.4.1 CPA

The CPA result depicted in Figure 7 leads to the same conclusion that CPA can not make a correct guess value distinguishable from incorrect ones for ISAP ISAPRk procedure.

3.4.2 t-test and χ^2-test

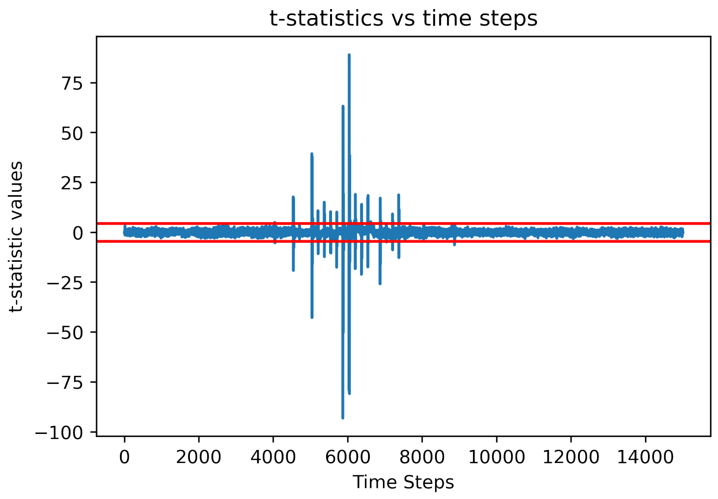


Figure 8: t-test result, the red line indicates the ± 4.5 threshold

For a further leakage analysis, we conducted t-test and χ^2-test on collected hardware implementations.

Welchs t-test is a statistical hypothesis test used to compare the means of two groups, especially when the two groups have unequal sample sizes and variances. In terms of side-channel analysis, we can divide the power traces into two groups according to the dierence in intermediate values.

More precisely, when the private key is xed, we can divide the power traces of by a bit of selected intermediate SBox output bits. Here we select the second bit for illustration.

As shown in Figure 8, both t-test and χ^2-test are able to detect obvious leakage on ISAP hardware implementation.

The t-test and χ^2-test shows tremendous leakage but the CPA fails to distinguish correct state guesses. For CPA, the correct guess and incorrect guess plot are either exactly the same or symmetric against x-axis. And t-test result shown in Figure 8 looks like amplification of y-scale of figure 7. It is bizarre that CPA result and t-test result look the same shape. We leave the explanation of this to the next section.

4 Results

4.1 Analysis

The anomaly of CPA results and t-test drove us into reviewing the ISAP permutation construction. The data to be absorbed Y_i each permutation is a single bit, thus only affecting a single bit of total 320 bit state. The linearity of SBox resides in operations of bits of the same position in five state variables. When the five bits are all XORed with associate data Y, a side-channel analysis could take place because all the bits of Y will be spread through SBox output. But in our selected ISAP's scenario, only first bit is XORed. This greatly reduces the leakage induced by SBox. When the attacker chooses a five-bit state initial state to guess, the SBox output is only determined by the bit to be absorbed.

Let x_i be bits of initial state to be guessed, and y be the bit to be absorbed, b_i be the SBox output bits. When x_i is fixed, whether our guess is correct or not, the b_i is only determined by y. The $b_i = y \oplus c_i^1 + c_i^2$ where c_i^1 and c_i^2 are fixed combination of x_0 to x_5. When conducting side-channel analysis on b_i, we're actually conducting on single-bit XOR. The non-linearity of SBox is significantly reduced by only absorbing a single bit each permutation. Any leakage on b_i actually comes from bit y rather than initial state x_i. For a single bit absorption scenario, there is no exploitable non-linearity of ISAP.

CPA results of correct guessing value and incorrect ones are the same because the intermediate values of each guess, which only determined by nonce bit, are either the same or opposite. Traces divided by the selected intermediate bit are also actually divided by the nonce bit, thus showing tremendous leakage.

4.2 Conclusion

In this report we have shown that ISAP is resilient against DPA-based Side-Channel Analysis. The attacker is unable to find a proper attack target in all components of ISAP. We further show that the leakage-resilience critically depends on the choice of r_B in the ISAPRk component. r_B indicates how many bits of data to be absorbed in the permutation. A small choice of r_B will lead the power leakage of master key to be covered by the leakage of publicly visible data. As a conclusion, for Ascon-based ISAP, the data to be absorbed has to be longer than 64 bit to conduct side-channel analysis.As a result, the current ISAP algorithm does not present any side-channel leakage with regard to the master key.

On the Side Channel Leakage Assessment of First-Order Masked GIFT-COFB

Xiangjun Lu[1], Shipei Qu[1], Tengfei Wang[1], Pei Cao[1]

School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China

1 Introduction

1.1 Background

GIFT-COFB is an Authenticated Encryption with Associated Data (AEAD) scheme, based on the GIFT lightweight block cipher and the COFB lightweight AEAD operating mode[BCI+20]. It has been selected as one of the finalists in the NIST lightweight cryptography standardization process. The side channel analysis of the native GIFT-COFB is carried out in [HBB20], while its mask-protected implementation has not yet been explored.

In power side-channel analysis, the attacker tries to recover secret information from the hardware running the cryptographic algorithm by recording the power consumption traces. In order to protect cryptographic algorithms from such attacks, it is often implemented with boolean masks to hide the real secret information.

In this report, we will perform a side-channel leakage assessment against GIFT-COFB with first-order boolean masking in both software and hardware implementations. The collected power traces are going through leakage detection and attack attempts to investigate the performance of the power side-channel resilience of GIFT-COFB.

1.2 Our Work and Results Overview

Our work in this report and the results of the side-channel leakage assessment on firstorder masked GIFT-COFB can be summarized as follows.

- We collected two trace sets from the given software and hardware implementations of GIFT-COFB on an MCU and a side-channel attack evaluation board.

- We performed Welch's t-test [BCD+13] and χ^2-test [MRSS18] to evaluate the power leakage of GIFT-COFB. We tried to recover the private keys of GIFT-COFB by correlational power attack (CPA).

- χ^2-test applied on the power traces from the given hardware implementations shows a slight potential power leakage from the input nonce. However, such leakage is missing in Welch's t-tests or χ^2-test on software implementations.

- CPA attack cannot recover the private key bytes under the given implementations.

2 Assessment Strategy

Our assessment strategy on the given GIFT-COFB implementations can be summarized as the following three phases:

1. Analysis of the target

The initial phase of GIFT-COFB is shown in Figure 1. We consider the known-plaintext attack scenario, which is a common assumption in side-channel attacks. Then we take the first block encryption component as the target of the attack, which is a GIFT-128 cipher.

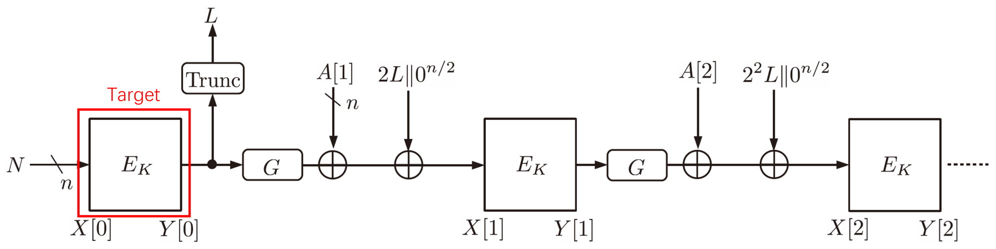


Figure 1: Analysis target

Next, we choose the output of *SubCells* operation in the second round as the intermediate value. The reason for choosing this intermediate value is that it has both a direct correspondence with the round key and a high degree of non-linearity.

Let's take the notation in the GIFT-COFB specification[BCI+20]. Specifically, the 128-bit secret key is loaded into the key state KS partitioned into 8 16-bit words:

$$KS = \begin{bmatrix} W_0 & \| & W_1 \\ W_2 & \| & W_3 \\ W_4 & \| & W_5 \\ W_6 & \| & W_7 \end{bmatrix} \leftarrow \begin{bmatrix} b_{127} & \cdots & b_{112} & \| & b_{111} & \cdots & b_{98} & b_{97} & b_{96} \\ b_{95} & \cdots & b_{80} & \| & b_{79} & \cdots & b_{66} & b_{65} & b_{64} \\ b_{63} & \cdots & b_{48} & \| & b_{47} & \cdots & b_{34} & b_{33} & b_{32} \\ b_{31} & \cdots & b_{16} & \| & b_{15} & \cdots & b_{2} & b_{1} & b_{0} \end{bmatrix} \tag{1}$$

And the cipher state S is expressed as 4 32-bit segments:

$$S = \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} \leftarrow \begin{bmatrix} b_{124} & \cdots & b_8 & b_4 & b_0 \\ b_{125} & \cdots & b_9 & b_5 & b_1 \\ b_{126} & \cdots & b_{10} & b_6 & b_2 \\ b_{127} & \cdots & b_{11} & b_7 & b_3 \end{bmatrix} \tag{2}$$

Suppose the cipher state before the *AddRoundKey* operation in the first round is $\{S_0^{'}, S_1^{'}, S_2^{'}, S_3^{'}\}$, which can be derived from the nonce N. Then *AddRoundKey* will update the cipher state with the round key and constant in the first round:

$$\begin{aligned} S_0^{''} &\leftarrow S_0^{'} \\ S_1^{''} &\leftarrow S_1^{'} \oplus W_6 || W_7, \\ S_2^{''} &\leftarrow S_2^{'} \oplus W_2 || W_3, \\ S_3^{''} &\leftarrow S_3^{'} \oplus \text{0x80000001} \end{aligned} \tag{3}$$

In order to perform side channel attacks such as CPA, we must be able to compute the corresponding intermediate value from parts of the key we guessed. In typical side-channel attacks (e.g. AES), one byte is often guessed and another byte is obtained as an intermediate value. However, this strategy does not work for the GIFT-128, because $S_1^{''}$ and $S_2^{''}$ affects multiple bytes in the output of the second round's *SubCells*:

$$\begin{aligned}
S_1 &\leftarrow S_1^{''} \oplus \left(S_0^{''} \& S_2^{''}\right) \\
S_0 &\leftarrow S_0^{''} \oplus \left(S_1 \& S_3^{''}\right) \\
S_2 &\leftarrow S_2^{''} \oplus (S_0 \mid S_1) \\
S_3 &\leftarrow S_3^{''} \oplus S_2 \\
S_1 &\leftarrow S_1 \oplus S_3 \\
S_3 &\leftarrow \sim S_3 \\
S_2 &\leftarrow S_2 \oplus (S_0 \& S_1) \\
\{S_0, S_1, S_2, S_3\} &\leftarrow \{S_3, S_1, S_2, S_0\}
\end{aligned} \tag{4}$$

Noting that the bit position of each byte does not change, so we can solve this problem by guessing 1 byte in $S_1^{''}$ and 1 byte in $S_2^{''}$, and calculate the corresponding 1 byte in the result. For example, if we choose the last byte of the output S_3 as the intermediate value, the calculation can be expressed as:

$$S_3[0] \leftarrow S_0^{''}[0] \oplus \left(\left(S_1^{''}[0] \oplus \left(S_0^{''}[0] \& S_2^{''}[0]\right)\right) \& S_3^{''}[0]\right)$$

where the index 0 indicates the position of the byte, $S_i^{''}$ can be obtained from Eq. 1 (2 bytes from $W_{2,6}/W_{3,7}$ is guessed). Based on the same principle, we can also use bit-level intermediate values, which can help to verify the leakage of side-channels more quickly.

2. Side-channel leakage detection

Next, we applied TVLA (Test Vector Leakage Assessment) to determine whether the collected power traces had noticeable plaintext or intermediate value leaks. Specifically, the main techniques used here are Welch's t-test and χ^2 test. They can roughly locate where in the traces the power leakage occurred.

3. Key recovery attack evaluation

Note that if there is power leakage detected in Phase 2, we can apply CPA here to reveal half of the key ($W_{2,3,6,7}$). The other half of the master key needs to attack the third round of *SubCells* with the same strategy based on the success of the first half of the key.

3 Experimental Setup

In this section, we will describe the details of power traces acquisition process.

3.1 Overall Procedure

The procedure of out power trace collection experiments is presented in Figure 2.

As shown in the figure, we first need to download the firmware which including the implementation of GIFT-COFB and our custom communication protocol into the device under evaluation. Then we connect the device to the host computer through a USB serial port so that we can invoke the cipher and record its input and output. Meanwhile, we use a high-precision electromagnetic probe to capture the electromagnetic power emitted from the device chip. The captured power is then transmitted to the oscilloscope to generate and display the waveform of electronic signals. With the help of the oscilloscope, we can acquire enough raw power traces of protected GIFT-COFB in the host computer for later assessment.

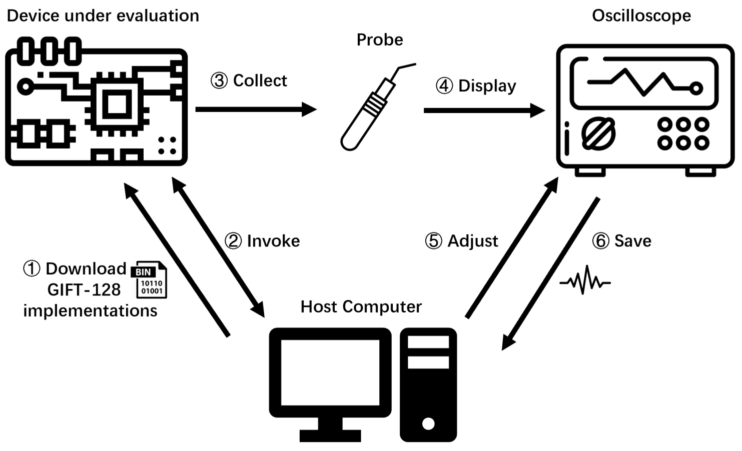


Figure 2: Overall procedure of power trace collection

3.2 Experimental Setting

3.2.1 Experimental environments

The details of devices and analyzing suites used for GIFT-COFB are presented in Table 1.

Table 1: Details of experimental environments

Type	Items	Details
Hardware platform	Target MCU	STM32F303RCT6
	Target evaluation board	Saseabo-giii(Kintex-7)
Measuring tools	High Precision EM probe	Langer RF-U 5-2
	Oscilloscope	Pico 3203D, LeCroy 610Zi
Sampling parameters	Sampling rate for MCU	125 MHz
	Sampling rate for FPGA	500 MHz
Random source	standard C library	`rand()`, `srand()` in stdlib.h

We assign GPIO_12 of STM32F303RCT6 (CN9 of Saseabo-gii) as the pin sending the trigger signals. The given software and hardware implementations of GIFT-COFB will be tested on STM32F303RCT6 and Saseabo-giii, respectively.

3.2.2 Input and output of GIFT-COFB

For the experiments of power trace collection on software implementation, the input of GIFT-COFB encryption consists of three parts: a 16-byte nonce, 16-byte associated data and 16-byte plaintext. The output consists of 16-byte ciphertext and a 16-byte authenticated tag. For the hardware implementation, it requires the input to be already masked data and thus twice as long as the original ones. The 16-byte encryption key is fixed throughout the collection. The specific information about the fixed input is shown in 2. All the fixed value are directly copied from the official test vectors provided in the implementer's code repository.

According to the analysis in 2, changing either the input nonce N or plaintext will change the intermediate values. Here we choose to alter the nonce in each encryption. Then the intermediate values will change under the same key, thereby generating different but related power consumption patterns. This allows us to perform CPA and other tests.

Table 2: Input details of GIFT-COFB

Implementation	Fixed Input	Value
Software	Master key	000102030405060708090A0B0C0D0E0F
	Plaintext	000102030405060708090A0B0C0D0E0F
	Associated data	000102030405060708090A0B0C0D0E0F
Hardware(masked)	Master key	B54F97F73F0716B75845D3D652C015A7
		FEA43B246C15EA6E619601E3FACC42A7
	Plaintext	C5F8D832CBF8D832
	Associated data	A25D267C615D267C

3.2.3 Trigger setting

Apart from the equipment mentioned in 2, another probe attached to the oscilloscope can receive trigger signals to help us locate the timing when GIFT-COFB is executed. Thus, we need to modify the original GIFT-COFB implementations so that they can control the corresponding pins of the device to send trigger signals to the oscilloscope.

For the software implementation, the codes to control the pin and send the trigger signals are inserted into prior and after the call to the first call to `giftb128_encrypt_block`, as shown in Figure 3.

```
int giftcofb_crypt(
...
    gift128_keyschedule(key, m_rkey.rkey, key_m);
    // Call trigger
    HAL_GPIO_TogglePin(Trigger_GPIO_PORT, Trigger_Pin);
    giftb128_encrypt_block(y, m_rkey.rkey, nonce);
    // Call trigger
    HAL_GPIO_TogglePin(Trigger_GPIO_PORT, Trigger_Pin);
```

Figure 3: Code snippet to set triggers in the software implementation

For the hardware implementation, we use a passive way to set the trigger signal, i.e. the algorithm will block until we supply a high level to a certain pin. The trigger is set as an external signal that enables the hardware to start executing the algorithm by pulling up for 1 clock cycle. This signal is also connected to the oscilloscope as a trigger control for the trace acquisition.

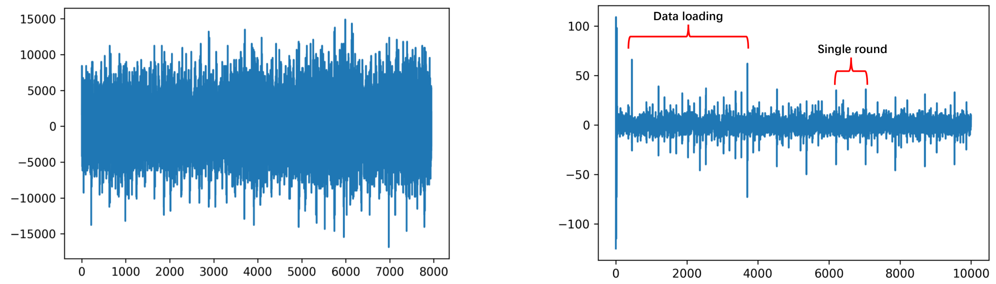
4 Description of Collected Raw Traces

We collected two sets of power traces, (S) and (H). (S) is acquired from the given software GIFT-COFB implementations under settings described in Section 3, and (H) is from the hardware implementation. Their basic information is presented in Table 3.

Table 3

Item	Software Implementation	Hardware Implementation
Trace set ID	S	H
Rounds contained	40	7
No. of traces	20,000	1,000,000
No. of points per trace	8,000	10,000
Precision	$-2^{15} \sim 2^{15}$	$-2^{7} \sim 2^{7}$
Sampling time	5h	12h

The sample plots of trace set (S) and (H) are presented in Figure 4. As seen from Figure 4a, we can easily distinguish the rounds in GIFT-128 encryption from (H).



(a) (S) (b) (H)

Figure 4: Sample graph of trace set (S) and (H)

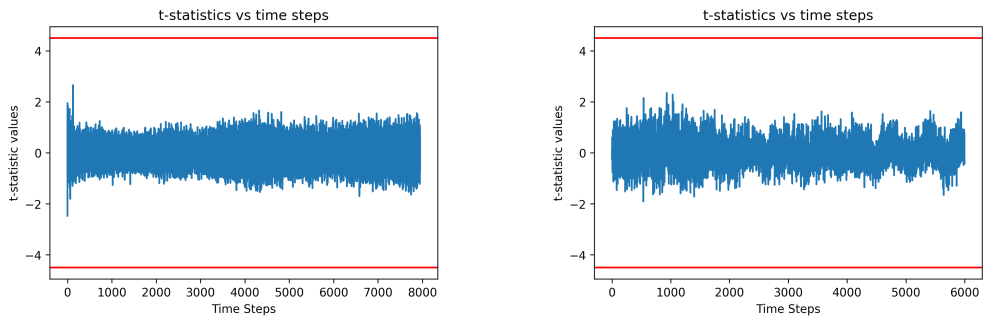
Then we can perform different tests mentioned in Section 2 on them to evaluate the power leakage of the given implementations.

5 Main Result

5.1 Welch's *t*-test

Welch's *t*-test is a statistical hypothesis test used to compare the means of two groups, especially when the two groups have unequal sample sizes and variances. In terms of side-channel analysis, we can divide the power traces into two groups according to the difference in intermediate values. More precisely, when the master key is fixed, we can divide the power traces of GIFT-COFB by the following two cases.

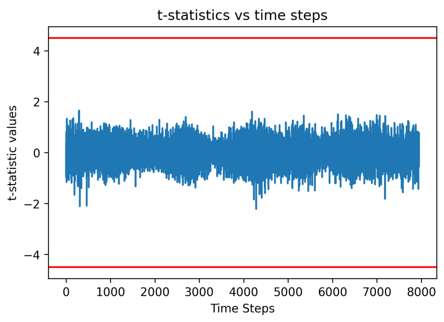
- Case(I): The last bit of the first byte of the input nonce is 0 or 1.
- Case(II): The last bit of the first byte of the intermediate value is 0 or 1.



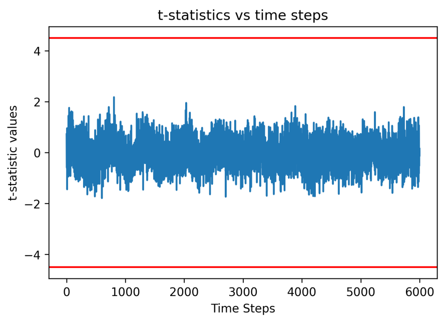
(a) (S) (b) (H)

Figure 5: Welch's t-test results of (S) and (H) (divided by Case (I))

The test results are shown in Figure 5 and Figure 6. We can see from the figure that the results failed to reach the threshold of the Welch's *t*-test for either the software or the hardware implementation, suggesting that no significant leakage information can be detected using this test approach.



(a) (S)



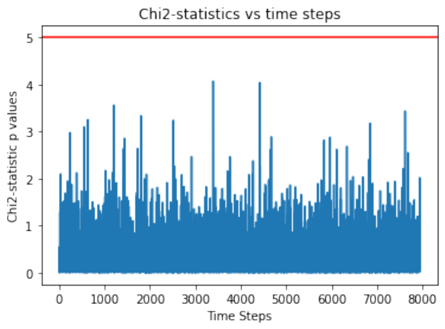
(b) (H)

Figure 6: Welch's t-test results of (S) and (H) (divided by Case (II))

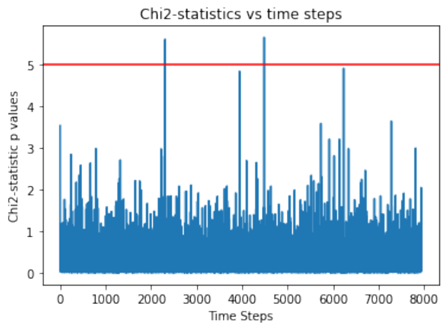
5.2 χ^2-test

χ^2-test is another statistical hypothesis test to determine whether there is a significant difference between the expected and observed frequencies, which is a natural complement to Welch's t-test for black box leakage detection, especially in the case of higher-order masked implementations. It can also test the null hypothesis of independence of a pair of random variables. Therefore, like t-test, we divide the power traces by the following two cases and observe their statistical differences.

- Case(I): The last bit of the first byte of the input nonce is 0 or 1.
- Case(II): The last bit of the first byte of the intermediate value is 0 or 1.



(a) (S)



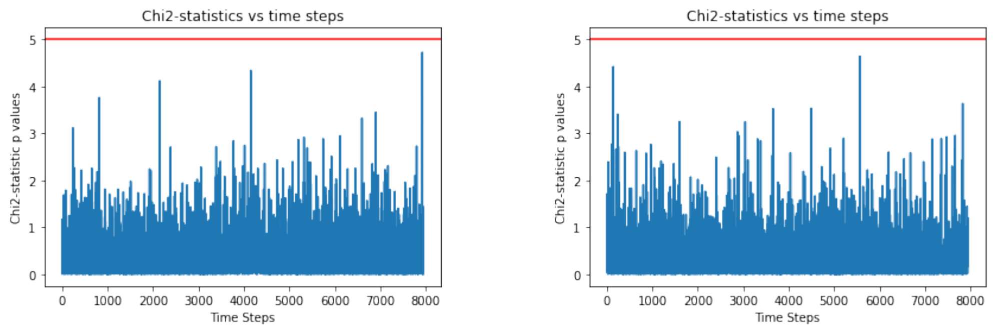
(b) (H)

Figure 7: χ^2-test results of (S) and (H) (divided by Case (I))

The test results are shown in Figure 7 and Figure 8. We can see from the figure that the results failed to reach the confidence level of theχ^2-test in 7a, but there is slight power leakage detected from (H) in Figure 7b when the traces are divided by nonce. However, when the traces are divided according to the intermediate values, χ^2-test cannot find statistically significant differences of two trace groups.

5.3 Correlational power attack (CPA)

CPA is an efficient side-channel analysis method to reveal the secret from power leakage of a cryptographic device. According to the analysis in Section 2, we will guess 1 bit (or 1



(a) (S) (b) (H)

Figure 8: χ^2-test results of (S) and (H) (divided by Case (II))

byte, using more computational resources) from two different subkeys $W_{2,6}/W_{3,7}$ at a time and compute the corresponding intermediate value. Taking the bit model for example, we can get a sequence of bits from the calculation of intermediate value. The correct guess will exhibit the greatest level of correlation between the bit sequence and real power trace, which indicates the correct subkey bit.

Table 4: CPA guess result of key bits in (S)

Target bit index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Target bit value (W2||W3)	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	1
Target bit value (W6||W7)	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	1	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	1
Real key rank(/4)	3	2	3	3	4	3	1	3	1	1	2	4	2	4	1	1	2	3	1	3	4	4	3	1	4	1	1	2	1	2	3	1
Best guess	01	10	01	10	00	00	00	11	00	00	01	10	00	00	00	11	10	11	00	01	10	10	10	00	10	00	00	01	01	01	00	11
Real key	00	00	00	00	01	11	00	00	00	00	00	00	01	11	00	11	00	00	00	00	01	11	11	00	00	00	00	00	01	11	11	11

Table 5: CPA guess result of key bits in (H)

Target bit index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Target bit value (W2||W3)	1	1	0	0	1	1	0	0	1	0	1	0	0	1	0	1	0	0	0	1	1	0	0	1	1	0	0	0	1	1	1	1
Target bit value (W6||W7)	0	1	0	1	1	1	1	1	1	1	1	0	1	1	0	0	0	1	0	1	1	0	0	1	1	1	0	1	1	1	0	1
Real key rank	4	2	3	2	4	3	3	1	2	2	4	4	1	3	1	1	1	2	1	2	1	2	3	1	1	2	2	1	3	3	3	2
Best guess	11	01	11	11	10	00	10	01	00	10	01	10	01	10	00	01	00	11	00	10	11	01	11	11	11	10	01	01	00	10	01	01
Real key	10	11	00	01	11	11	01	01	11	11	11	00	01	11	00	01	00	01	00	11	11	00	00	11	11	01	00	01	11	11	10	11

For trace set (S), we perform CPA on $W_{2,6}$ and $W_{3,7}$, including half of all key bits of GIFT-128. The CPA guess results for each bit is presented in Table 4.

Note that we are guessing 2 bits of 2 different subkeys each time, hence the guessed bits and real key bits are expressed as $(W2||W3)[i]||(W_6||W_7)[i]$, where i indicates the bit location from the lowest. In the result above, the average ranking of the correct key is 2.3125, which is similar to the theoretical result of 2.5 for any random guess. Furthermore, the success rate of key guessing over all bits is 34.375%, which is a slight increase compared to the random guess of 25%, but still far from restoring the real master key.

For trace set (H), we have the following CPA results in Table 5.

The hardware implementation results are similar to the previous ones overall. The average rank order of correct keys in the correlation results is 2.1875, and the percentage of correct guessed keys is 28.125%. In general, CPA fails to perform effective attacks on the given software and hardware implementations.

References

[BCD+13] Georg T. Becker, Jim Cooper, Elizabeth K. DeMulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, T. Kouzminov, Andrew J. Leiserson, Mark E. Marson, Pankaj Rohatgi, and Sami Saab. Test vector leakage assessment (tvla) methodology in practice. 2013.

[BCI+20] Subhadeep Banik, Avik Chakraborti, Akiko Inoue, Tetsu Iwata, Kazuhiko Minematsu, Mridul Nandi, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. Gift-cofb. Cryptology ePrint Archive, Paper 2020/738, 2020. https://eprint.iacr.org/2020/738.

[HBB20] Xiaolu Hou, Jakub Breier, and Shivam Bhasin. Dnfa: Differential no-fault analysis of bit permutation based ciphers assisted by side-channel. Cryptology ePrint Archive, Paper 2020/1554, 2020. https://eprint.iacr.org/2020/1554.

[MRSS18] Amir Moradi, Bastian Richter, Tobias Schneider, and François-Xavier Standaert. Leakage detection with the x2-test. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):209–237, Feb. 2018.

On the Side-channel Leakage Assessment of Ascon with Boolean Masking

Hongyi Zhang[1], Pei Cao[1]

School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China

1 Introduction

In the report, we will make a side-channel leakage assessment for Ascon with second-order boolean masking in software implementation(STM32F303) and first-order boolean masking in hardware implementation(Sakura-X). With collected power traces, the report will show the capability of counter side-channel analysis of Ascon through three tests, including Welch's t-test, χ^2-test, and correlation power analysis (CPA).

2 Our Work and Results Overview

In this report, our work and the assessment results of the side-channel leakage assessment on Ascon can be concluded as follows:

- We collected two trace sets from the given software and hardware implementations of Ascon on an STM32F303 MCU and a SAKURA-X evaluation board
- We performed Welch's t-test and χ^2-test to evaluate the power leakage condition of Ascon. Also, we tried to recover the private keys of Ascon by CPA
- Welch's t-test and χ^2-test did **not** show obvious leakage of intermediate value
- CPA **cannot** recover the private key bits in the software and hardware implementations when protection is applied to Ascon

3 Assessment Strategy

There are three phases in the assessment strategy on the given Ascon implementations:
Phase 1: Determine the intermediate value for analysis.
As far as we know, unprotected Ascon has been shown to be insecure under CPA [SD17]. Since we know the contents of the state at initialization, except for the key part. And we can vary the nonce each run, we pick the end of the first round of the initialization phase as our point of analysis (see figure 1). Specifically, we chose the output of the first round of permutation Keccak-p as the intermediate value:

$$y0 = k(m' + 1) + m \tag{1}$$

where $y0$ is the output of the non-linear S-box, k is one bit of the key, m' and m are related to the variable nonce.
Phase 2: Discern possible power leakage.

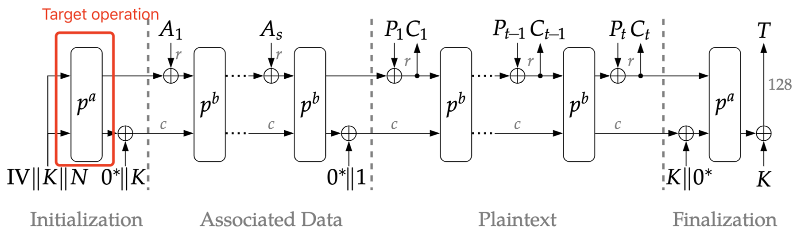


Figure 1: Encryption of Ascon

Next, we need to determine if there is power leakage in the process of Ascon encryption. As mentioned before, we assess the power traces with Welch's t-test and χ^2 test, which can help us discover the possible leakage points in the power trace. These leakage points in turn help us to find the corresponding operation that cause the leakage.
Phase 3: Recover the secret key bits.
If we detect the possible power leakage in Phase 2, then we can apply CPA to the Ascon to recover the key bits. In the process of CPA, we would foucs on the first three bits of the private key, which should be easily obtained if there is leakage in Ascon's encrpytion.

4 Assessment on the EM traces of software implementation

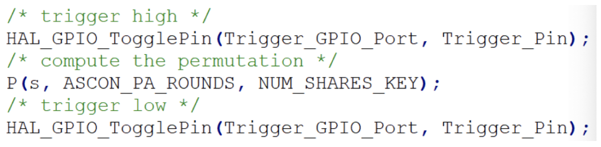
4.1 Experiment Procedure

First, we download the Ascon's firmware of the C implementation into the STM32F303's flash memory. Then we connect the STM32F303 chip to the host computer through a USB serial port to execute the algorithm and record the input and output data.At the same time, we collect the electromagnetic power traces of the chip with a high-precision electromagnetic probe. And the sampled traces are recorded by the Pico-3203D oscilloscope in the form of .trs files. After repeatedly collected a large number of traces, we can have a power trace set of the masked Ascon for the assessment. In the experiment, the software code applied to STM32F303 can be found on github(https://github.com/ascon/simpleserial-ascon/releases/tag/v1.2.6).

Table 1: Details of experimental environments for software implementation.

Items	Details
Target MCU	STM32F303RCT6
EM probe	Langer RF-U 5-2
Oscilloscope	Pico 3203D
Baud rate (USB Serial Port)	115200 bps
Sampling rate	62.5MS/s
Amplifier	Mini-Circuits ZKL-1R5+
Ascon Code Version	protected_bi32_armv6

Trigger location. When sampling traces, we use a trigger signal to locate the timing when the target operation (the permutation Keccak-p in the initialization phase) is executed. Therefore, we need to modify the original Ascon implementations so that they can control the corresponding pins of the device to send trigger signals at sepcific time point. To achieve this function, we insert the controlling soruce codes into the 'ascon_initaead' function, specifically, before and after the call of the 'P' function (see figure 2). To cover larger parts of the implementation, the number of rounds have been reduced to 2 rounds for PA and PB.



```
/* trigger high */
HAL_GPIO_TogglePin(Trigger_GPIO_Port, Trigger_Pin);
/* compute the permutation */
P(s, ASCON_PA_ROUNDS, NUM_SHARES_KEY);
/* trigger low */
HAL_GPIO_TogglePin(Trigger_GPIO_Port, Trigger_Pin);
```

Figure 2: Code snippet to set trigger in the software implementation

Input data of Ascon. During the experiments of EM trace collection, the input of Ascon encryption consists of three parts: a 16-byte nonce, 16-byte associated data and 16-byte plaintext. The output is 32-byte, including a 16-byte ciphertext and a 16-byte authenticated tag. The 16-byte encryption key is fixed throughout the collection. The detailed information about the fixed input is shown in table 2. Since changing solely the input nonce will change the intermediate values, we choose to alter the nonce in each encryption. Then the intermediate values will change under the same key, thereby generating different but related power consumption patterns. This allows us to perform CPA and other tests.

Table 2: Details of input for software Ascon implementation.

Items	Details
Key	000102030405060708090A0B0C0D0E0F
Plaintext	000102030405060708090A0B0C0D0E0F
Associated data	000102030405060708090A0B0C0D0E0F
Nonce	random

Trace information. The basic information of collected traces is presented in Table 3.

Table 3: Basic information of the collected EM traces for software Ascon implementation.

Items	Details
No. of traces	60000
No. of points per trace	80000
Precision	$-2^{15} \sim 2^{15}$
Sampling time	4 hours

4.2 Result of Welch's t-test

We use Welch's t-test, which is used to compare the means of two sample groups, to examine if there is any leakage points in the power trace. To do the t-test, we compart the power traces into two groups according to the difference in their intermediate values. As the private key is fixed, we can divide the power traces of Ascon by the intermediate value $y0$ in equation 1, which is determined by the first bit of the secret key. The test results are shown in figure 3.

4.3 Result of χ^2-test

χ^2-test is a statistical hypothesis test to determine whether there is a significant difference between the expected and observed frequencies. It can also test the null hypothesis of

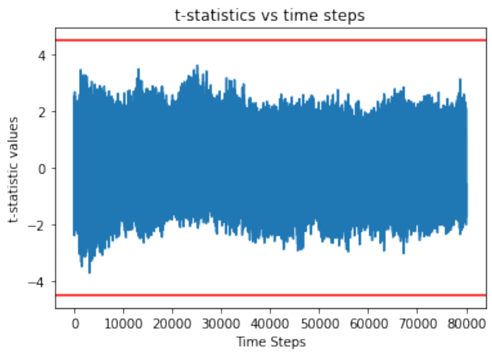


Figure 3: Results of Welch's t-test on Ascon software implementation

independence of a pair of random variables. Therefore, like t-test, we need to divide the power traces of Ascon by the intermediate value $y0$ in equation 1 and observe their statistical differences.

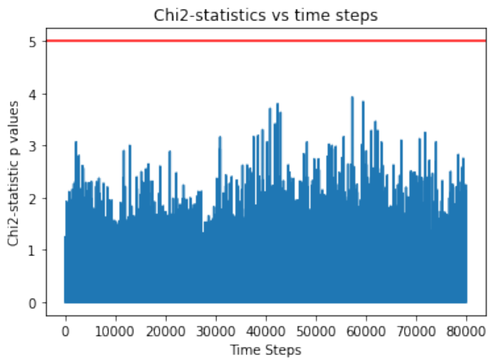


Figure 4: Result of χ^2-test on Ascon software implementation

4.4 CPA results for software power traces

We try to recover the first three bits of the key by using CPA(Correlation Power Analysis), which should be 000. In this way, we get the result as below, the figure shows the leakage situation at different sample points.

And best 8 keys guessed by CPA are shown in Table 4:

We can see the best key is 4 and 5, which are not the correct ones. Therefore, the CPA against Ascon power traces of software implementation is not successful, showing that the Ascon can resist the side-channel attacks.

Table 4: Best 8 keys guessed by CPA.

Key Rank	Key	Correlation Value	Leak Position
0	4(100)	0.014632203153923171	17218
1	5(101)	0.014632203153923171	17218
2	0(000)	0.014263695444477729	39050
3	1(001)	0.014263695444477729	39050
4	6(110)	0.013485215252643857	47599
5	7(111)	0.013485215252643857	47599
6	2(010)	0.012931953494060296	40501
7	3(011)	0.012931953494060296	40501

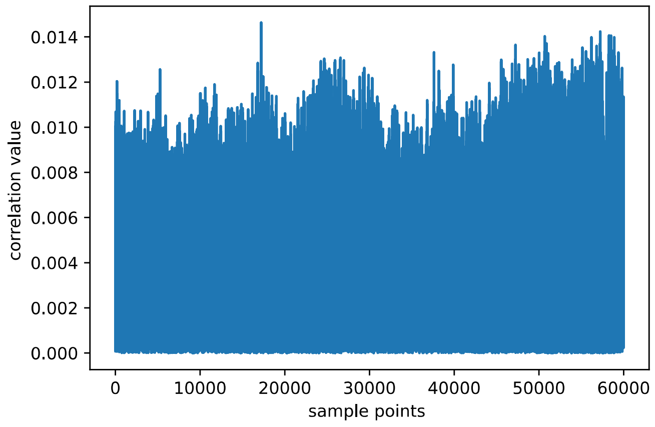


Figure 5: CPA result for Ascon using software power traces

5 Assessment on the power traces of hardware implementation

5.1 Experimental Setting

We first need to download the firmware of Ascon into the SAKURA-X. Then we connect the device to the host computer through a USB serial port so that we can execute the cipher and record its input and output. The captured power comsumption is then transmitted to the oscilloscope to generate and display the waveform of electronic signals. With the help of the oscilloscope, we can acquire enough raw power traces of protected Ascon in the host computer for later assessment. The source code of hardware implementation can be found online(https://cryptography.gmu.edu/athena/LWC/LWC_Finalists_protected_HW_implementations.html).

Input data of Ascon. During the experiments of power trace collection, the input of Ascon encryption consists of four parts (see table 6). Only nonce is variable, the other inputs (i.e., key, plaintext, and associated data) are fixed.

Trace information. The basic information of collected traces is presented in Table 7.

Table 5: Details of experimental environments for hardware implementation.

Items	Details
Target platform	SAKURA-X (with Xilinx Kintex-7 FPGA)
Oscilloscope	LeCroy 610Zi
Sampling rate	1GS/s
Ascon code version	ASCON_HPC2(first order boolean mask)

Table 6: Details of input for hardware Ascon implementation.

Items	Details
Key	D90E654D39818255180DD3DCA9FAEB4B
Plaintext	/
Associated data	81B2D700
Nonce	random

Table 7: Basic information of the collected traces for hardware Ascon implementation.

Items	Details
No. of traces	1000000
No. of points per trace	10000
Precision	$-2^7 \sim 2^7$
Sampling time	8 hours

5.2 Result of Welch's t-test

The result is shown in figure 6

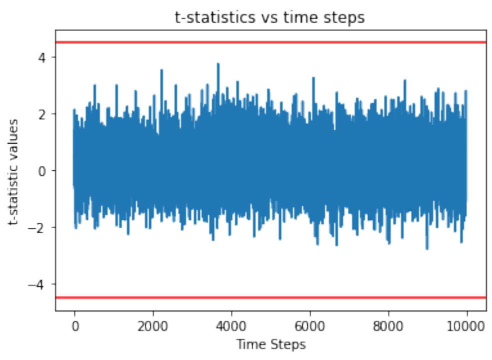


Figure 6: Results of Welch's t-test on Ascon hardware implementation

5.3 Result of χ^2-test

The result of χ^2-test is as below:

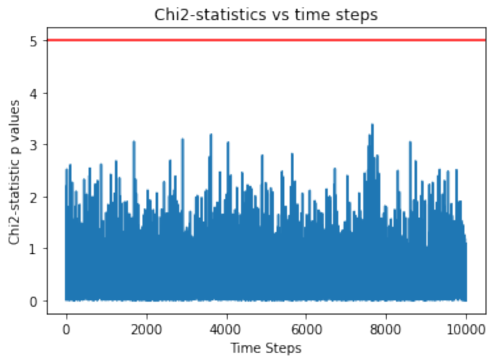


Figure 7: Results of χ^2-test on Ascon hardware implementation

5.4 CPA results for hardware power traces

We try to recover the first three bit of the key by using CPA(Correlation Power Analysis), which should be 000. And we get the result as below, the figure 8 shows the leakage situation at different sample points. We can see the best guessing is 110, which is far from being the right answer 000.

Best 8 guessed keys are shown in Table 8 below:

Table 8: Best 8 keys guessed by CPA.

Key Rank	Key	Correlation Value	Leak Position
Rank0	key 6(110)	0.049118043345022504	9823
Rank1	key 7(111)	0.049118043345022504	9823
Rank2	key 2(010)	0.04863842222881651	7496
Rank3	key 3(011)	0.04863842222881651	7496
Rank4	key 0(000)	0.045888110508008735	7497
Rank5	key 1(001)	0.045888110508008735	7497
Rank6	key 4(100)	0.04552392802497998	7496
Rank7	key 5(101)	0.04552392802497998	7496

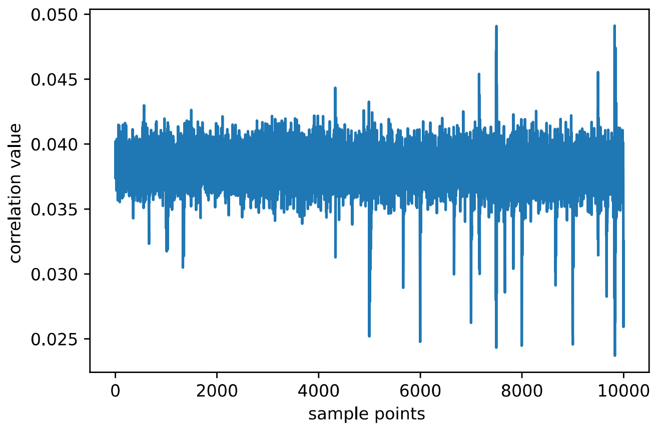


Figure 8: CPA result for Ascon using hardware power traces

References

[SD17] Niels Samwel and Joan Daemen. DPA on hardware implementations of ascon and keyak. In *Proceedings of the Computing Frontiers Conference, CF'17, Siena, Italy, May 15-17, 2017*, pages 415–424. ACM, 2017.