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 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.

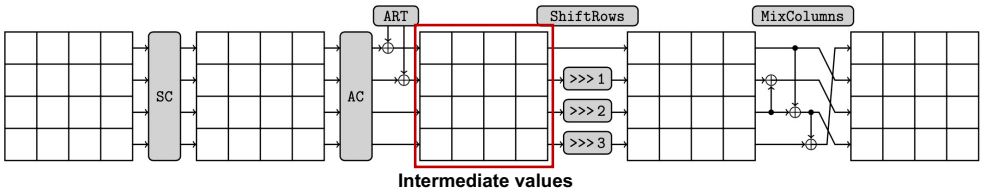


Figure 1: One encryption round of `Skinny-128-384+` (from Romulus documentation)

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.

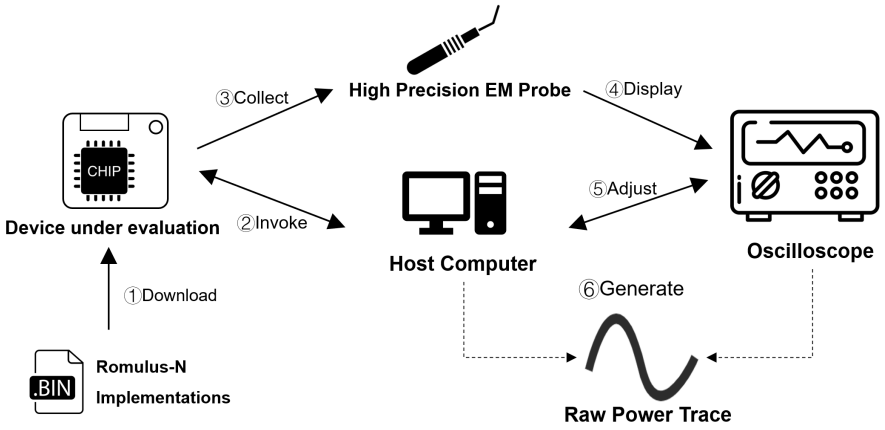


Figure 2: Overall procedure of power trace collection

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.

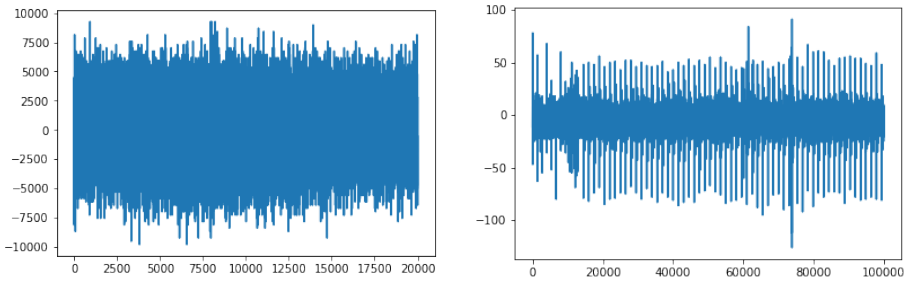
4 Description of Collected Raw Traces

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

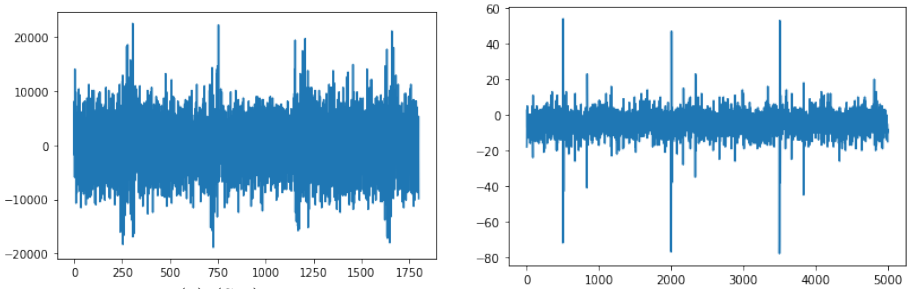
The sample graphs of trace set (S-1) and (H-1) are presented in Figure 5. As seen from Figure 5(b), we can easily distinguish 40 rounds in Skinny encryption from (H-1).



(a) (S-1) (b) (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).



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

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

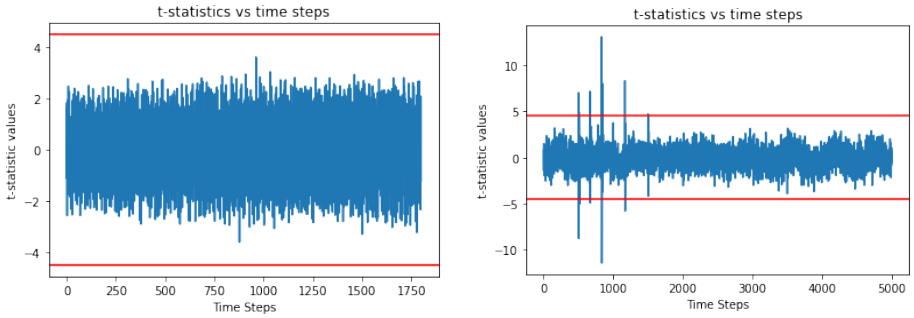
Since (S-2) and (H-2) has better sampling quality and smaller dimensions, all the tests below will be performed on them.

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.

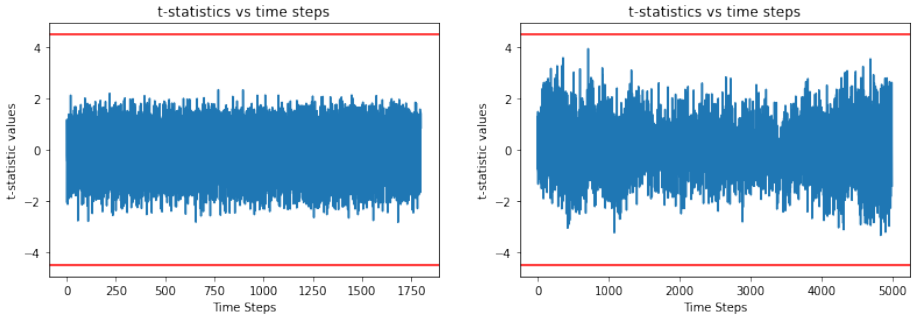
More precisely, when the private key is fixed, we can divide the power traces of Romulus-N by the following two cases.

- **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))

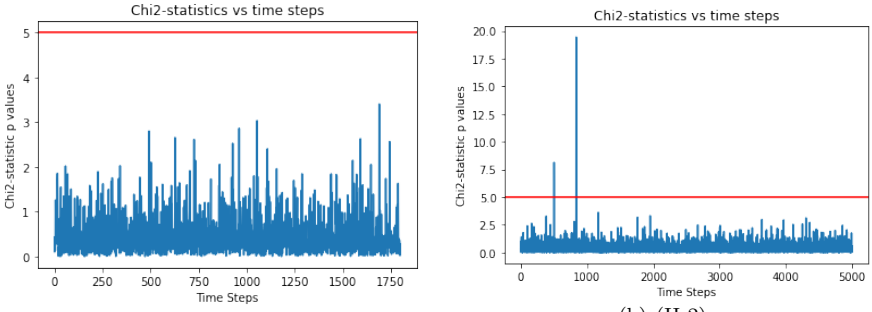
The test results are shown in Figure 7 and 8. We can see that the results of (S-2) fail to achieve the confidence level of Welch's t-test, but there is significant power leakage detected from (H-2) in Figure 7(b) when the traces are divided by nonce. Moreover, the overall range causing leakage matches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

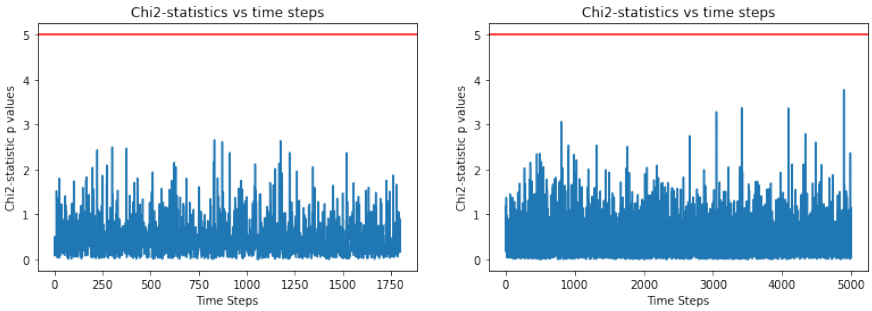
- **Case (A)**: The last bit of the first byte of the input nonce is 0 or 1.
- **Case (B)**: The last bit of the first byte of the intermediate value is 0 or 1.

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))

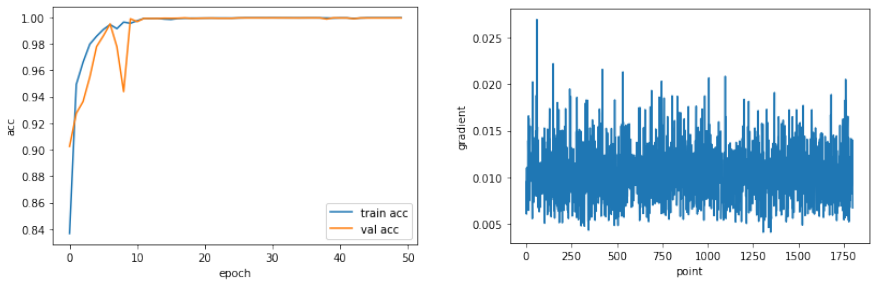


(a) (S-2) (b) (H-2)

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.

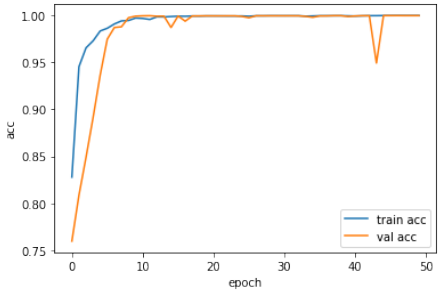


(a) accuracy of the CNN model (b) overall accumulated gradients of (S-2)

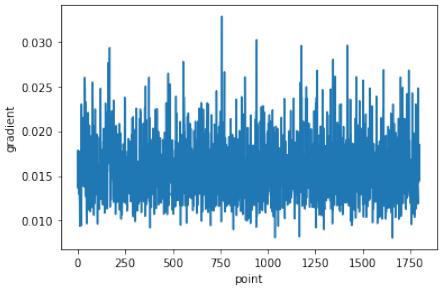
Figure 11: Assessment results of DL-LA on (S-2) (divided by Case (A))

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.

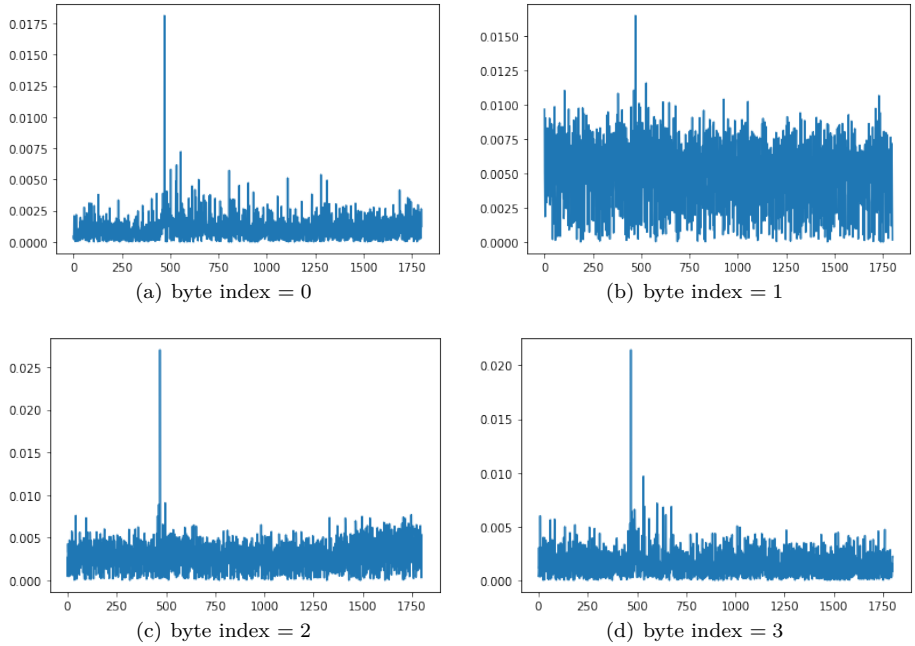


Figure 13: CPA correlation result of (S-2)

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

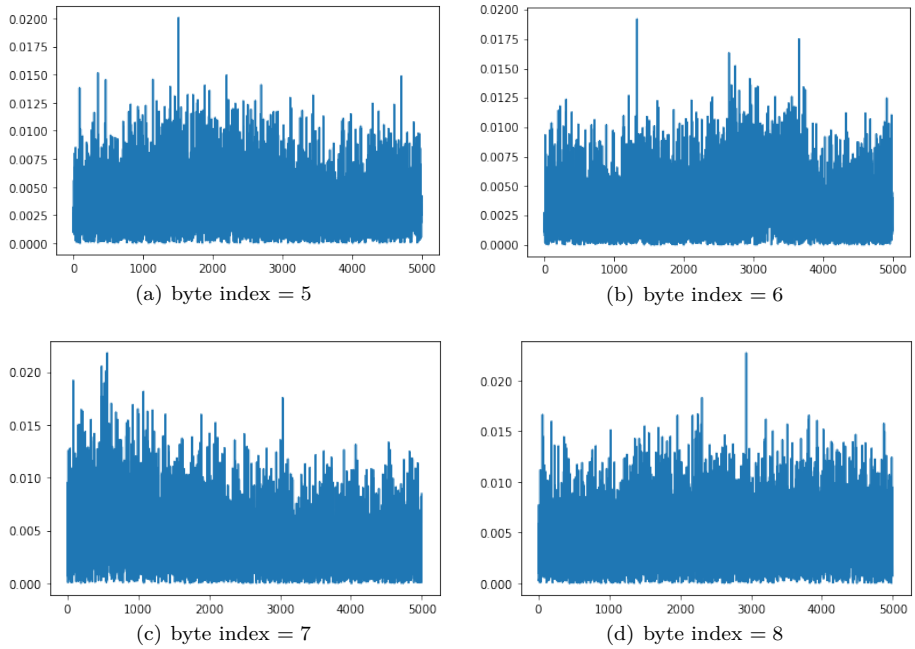


Figure 14: CPA correlation result of (H-2)

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.