Leakage Assessment Report for First-order Masked Romulus

Shuohang Peng Jiangxue Liu Bohan Yang Wenping Zhu Leibo Liu

September 14, 2022

1.Target implementation

(a) Algorithm: **Romulus.**
(b) Team: **Alexandre Adomnicai.**
(c) Variant name: **Romulus-N ARMv7-M implementation (1st-order masking countermeasure)**
(d) URL: **https://github.com/aadomn/romulus_adomnicai**
(e) GitHub commit hash: **a1eb0bd0f8473db41a713828001be8cc80f95da3**
(f) Protection method: **Boolean masking.**
(g) Protection order: **1.**

2.Experimental setup

(a) Measurement platform and device-under-evaluation: **ChipWhisperer CW308 with STM32F303 UFO target.**
(b) Description of measurements: **The design-under-evaluation power consumption is measured with the voltage drop across the on-board 12 Ω shunt resistor.**
(c) Usage of bandwidth limiters, filters, amplifiers, etc. and their specification: **N/A.**
(d) Frequency of operation: **8 MHz.**
(e) Oscilloscope and its major characteristics: **Teledyne LeCroy WaveRunner 8404M with 4 GHz bandwidth was used to collect traces.**
(f) Sampling frequency and resolution: **Sampling rate of 25 MS/s and 8-bit sample resolution were used.**
(g) Are sampling clock and design-under-evaluation clock synchronized? **No.**

3.Leakage assessment characteristics

(a) Leakage assessment type: **Fixed message vs. random message t-test at first order and fixed key vs. random key t-test at first order.** [GGR11**]**
(b) Number of traces used: **100,000.**
(c) Data inputs and performed operations: **Tested operation is the crypto_aead_encrypt_shared/crypto_aead_decrypt_shared. Input test vectors are generated on PC and sent to the target board.**
(d) Source of pseudorandom inputs: **The rand( ) and srand( )(Generate random seed from PC) functions in C.**
(e) Trigger location relative to the execution start time of the algorithm: **Scope trigger is set before and after crypto_aead_encrypt_shared/crypto_aead_decrypt_shared.**
(f) Time required to collect data for a given leakage assessment: **About 3 hours.**
(g) Total time of the assessment: **About 3 hours.**

(h) Availability of raw measurement results: **Per request.**

(i) Fixed input and output of Romulus-N: **Shown in Table 1.**

Table 1: Input/output details of Romulus-N

Fixed input/output	Value
Key	000102030405060708090A0B0C0D0E0F
Nonce	000102030405060708090A0B0C0D0E0F
PT	00010203
AD	00010203
CT	2BB84CB9B5D2B0D7321FA329FC633B1289D6C6A5

4.Results of leakage assessment

(a) Graphs illustrating the obtained results: **For the fixed message vs. random message t-test, the result of protected encryption on 100,000 traces is shown in Figure 1(top: trigger and trace, bottom: t-test result). The result of protected decryption on 100,000 traces is shown in Figure 2.**

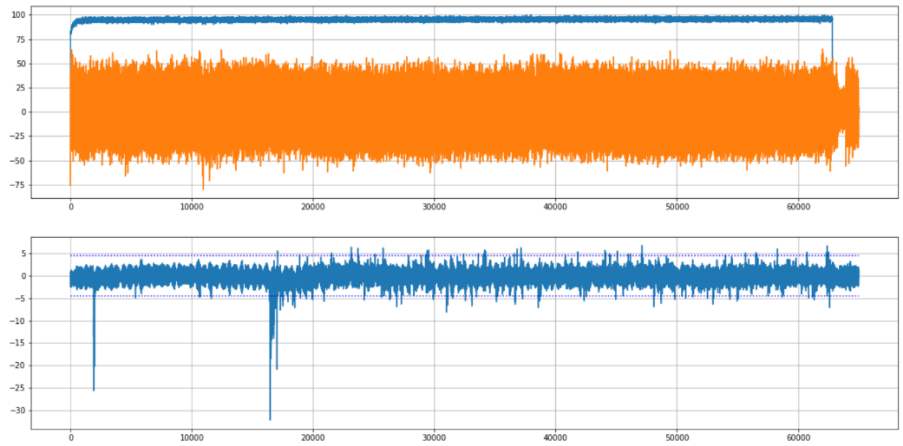

Figure 1: First-order t-test results of encryption with the fixed message vs. random message (100,000 traces).

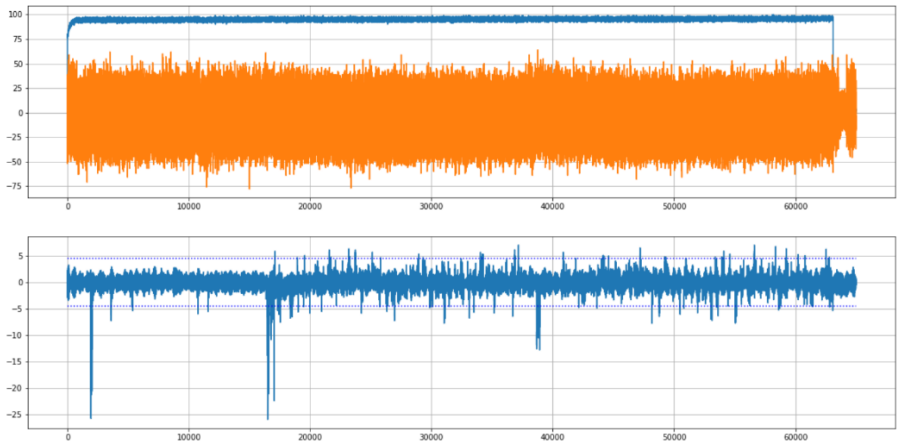

Figure 2: First-order t-test results of decryption with the fixed message vs. random message (100,000 traces).

For the fixed key vs. random key t-test, the result of protected encryption on 100,000 traces is shown in Figure 3. The result of protected decryption on 100,000 traces is shown in Figure 4.

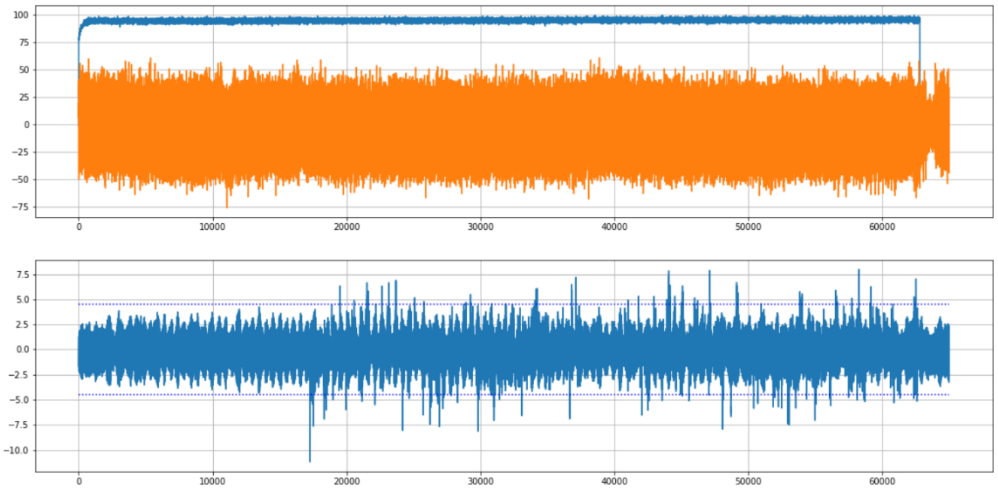

Figure 3: First-order t-test results of encryption with the fixed key vs. random key (100,000 traces).

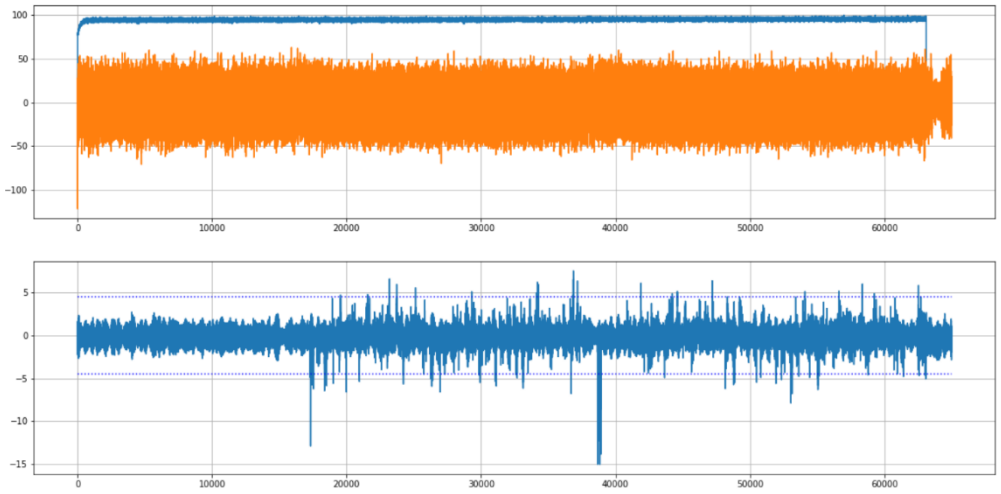

Figure 4: First-order t-test results of decryption with the fixed key vs. random key (100,000 traces).

(b) Leakage analysis: **It can be seen that there are leaks in both t-tests. For the fixed message vs. random message t-test, We marked the location of the two leakage spikes by the trigger, as shown in Figure 5.**

The first obvious leakage occurs when the state is XORed with ad in function romulusn_process_ad (the code shown in Figure 6), and we think the reason for the leak is that ad is not shared. The second obvious leakage is at the beginning of the skinny128_384_plus function, which requires ad-hoc analysis.

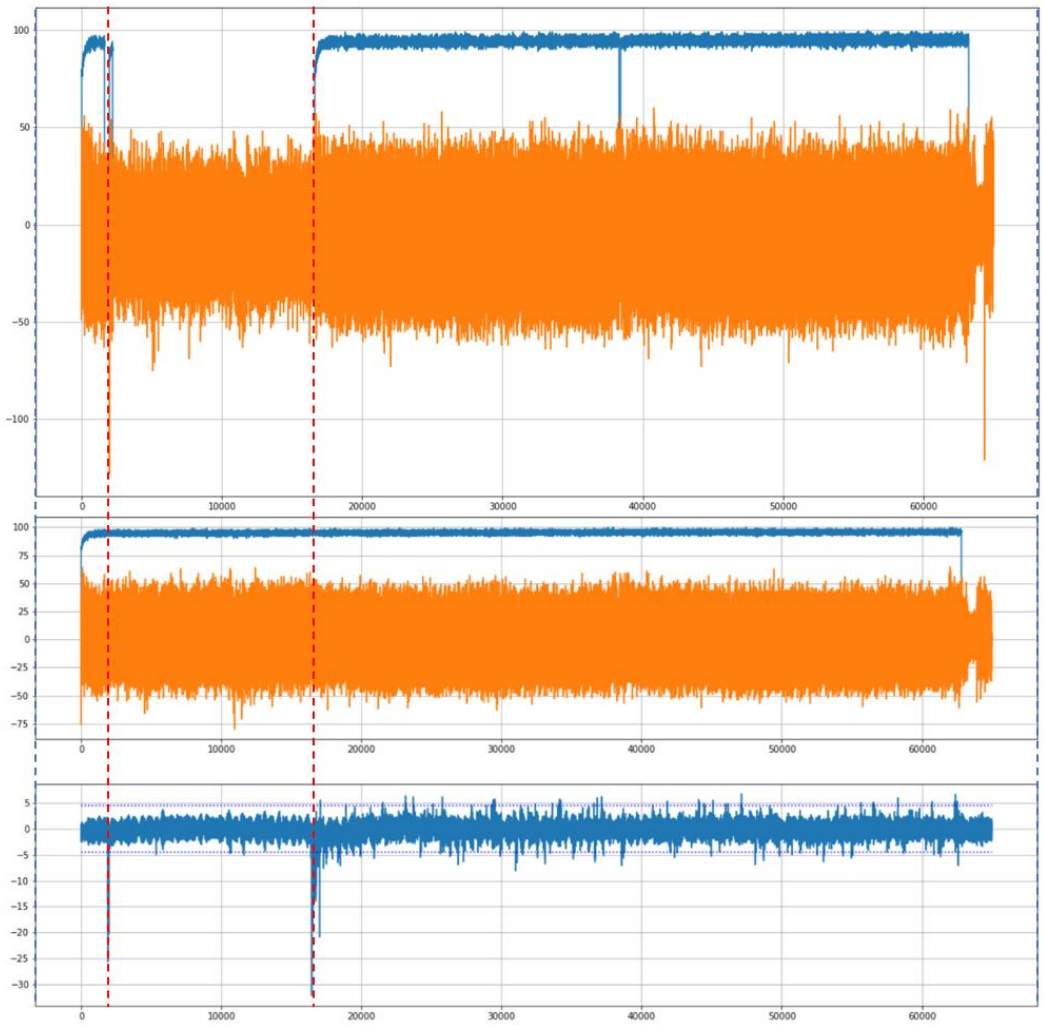

Figure 5: Leakage spikes marked with red lines

```
99              } else if (adlen == BLOCKBYTES) {    // Left-over complete single block
100                     XOR_BLOCK(state, state, ad);
101                     SET_DOMAIN(tk1, 0x18);
102             } else {      // Left-over partial single block
103                     trigger_high();
104                     for(i = 0; i < (int)adlen; i++)
105                             state[i] ^= ad[i];
106                     state[15] ^= adlen;
107                     SET_DOMAIN(tk1, 0x1A);
108                     trigger_low();
109             }
110              tk_schedule_123(rtk, rtk_m, rtk1, tk1, npub, k, k_m);
111              trigger_high();
112              skinny128_384_plus(state, state_m, state, state_m, rtk, rtk_m, rtk1);
113              trigger_low();
114     }
115 }
```

Figure 6: Code corresponding to leakage spikes

References

[GGR11] Tunstall M, Goodwill G. Applying TVLA to public key cryptographic algorithms[J]. Cryptology ePrint Archive, 2016.