Leakage Assessment Report for First-order Masked GIFT-COFB

Shuohang Peng Jiangxue Liu Bohan Yang Wenping Zhu Leibo Liu

September 14, 2022

1.Target implementation

(a) Algorithm: **GIFT-COFB.**
(b) Team: **Alexandre Adomnicai.**
(c) Variant name: **First-order masked ARMv7-M implementations of GIFT-COFB AEAD scheme**
(d) URL: https://github.com/aadomn/giftcofb_adomnicai
(e) GitHub commit hash: **6595c7373d40602e1d6c00f6f73f435d7869efac**
(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 90 minutes.**
(g) Total time of the assessment: **About 90 minutes.**

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

4.Results of leakage assessment

(a) **Since the type of implementation is not specified in the documentation, we first perform the fixed message vs. random message t-test on the entire implementation. 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.**

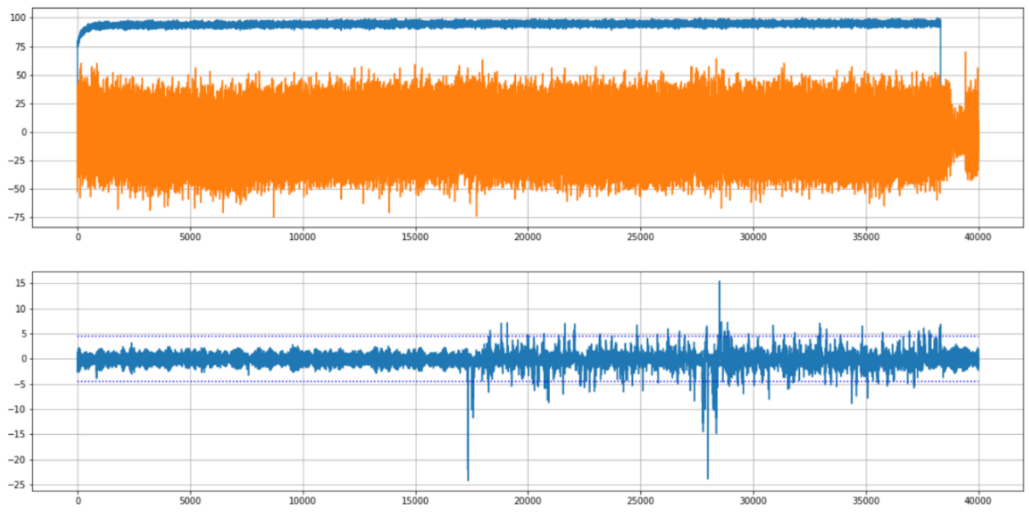


Figure 1: First-order t-test results of encryption with the fixed message vs. random message (100,000 traces).

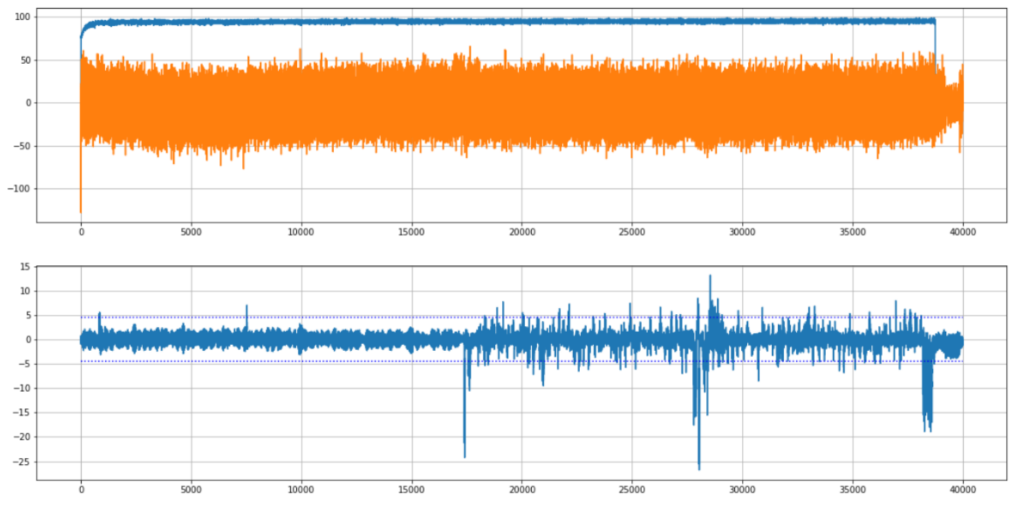
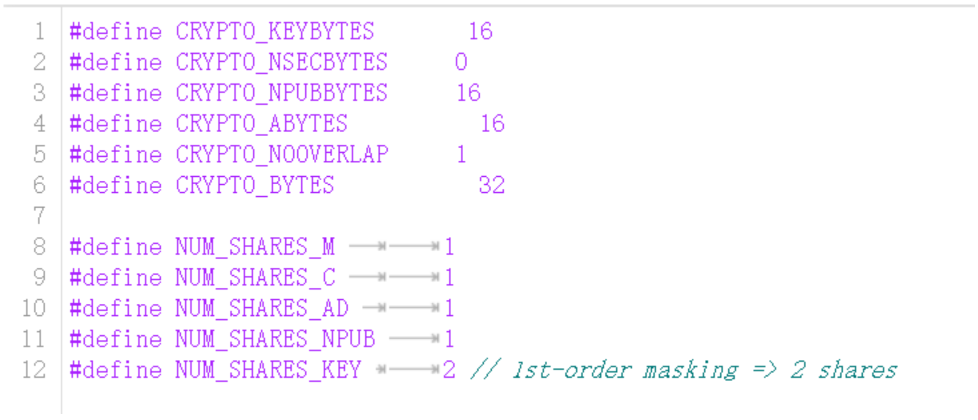


Figure 2: First-order t-test results of decryption with the fixed message vs. random message (100,000 traces).

It can be seen that there are leaks in the implementation, and it mainly exists inthe second half of the entire encryption and decryption process(This half of the algorithm mainly performs encryption and decryption operations.). In the

"api.h", we find they only divide the key into shares, thus we think this implementation may be a level implementation, which resists DPA in the subkey generation stage and resists SPA in the encryption and decryption stage[M20][PSV15].



```
#define CRYPTO_KEYBYTES       16
#define CRYPTO_NSECBYTES     0
#define CRYPTO_NPUBBYTES     16
#define CRYPTO_ABYTES          16
#define CRYPTO_NOOVERLAP     1
#define CRYPTO_BYTES           32

#define NUM_SHARES_M       1
#define NUM_SHARES_C       1
#define NUM_SHARES_AD      1
#define NUM_SHARES_NPUB    1
#define NUM_SHARES_KEY     2 // 1st-order masking => 2 shares
```

Figure 3: Settings of shares number in api.h

Therefore we do the fixed key vs. random key t-test to check for leaks. The result of protected encryption on 1M traces is shown in Figure 4. The result of protected decryption on 1M traces is shown in Figure 5.

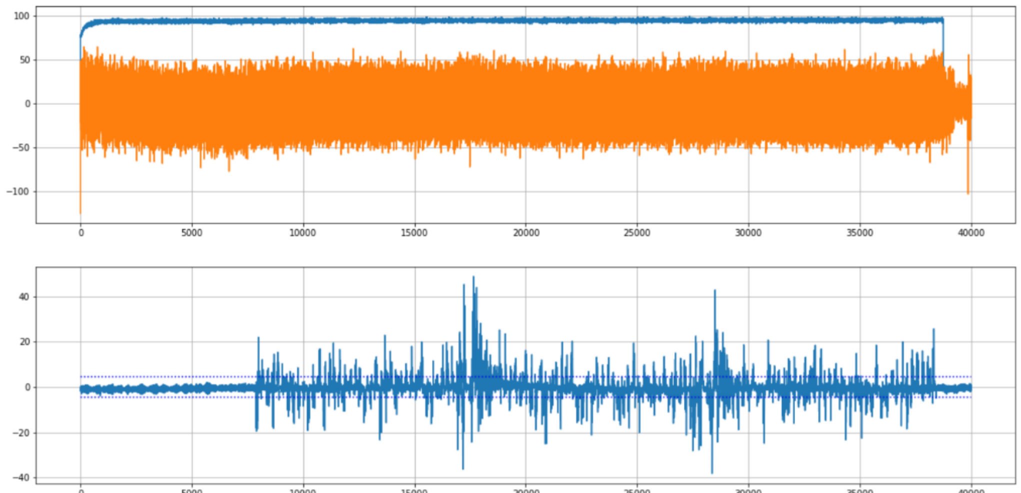


Figure 4: First-order t-test results of encryption with the fixed key vs. random key (1M traces).

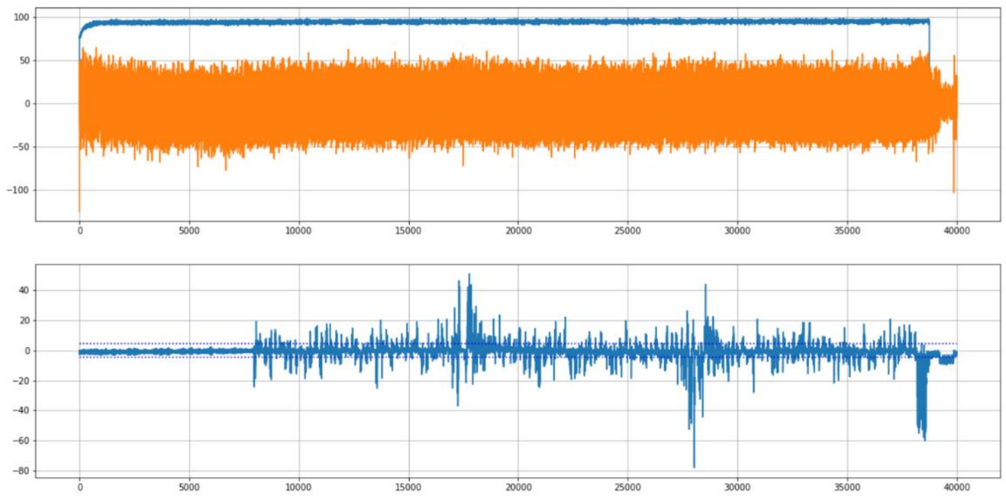


Figure 5: First-order t-test results of decryption with the fixed key vs. random key (1M traces).

(b) Leakage analysis: **It can be seen that there are leaks in both t-tests, and we have located the specific locations where these leaks occur. For the fixed message vs. random message t-test, we placed the trigger on both sides of the code containing m (decryption corresponds to c) and ad, as shown by the red dashed lines in Figures 6 and 7 below. It can be seen that the leakage mainly occurs in this phase.**

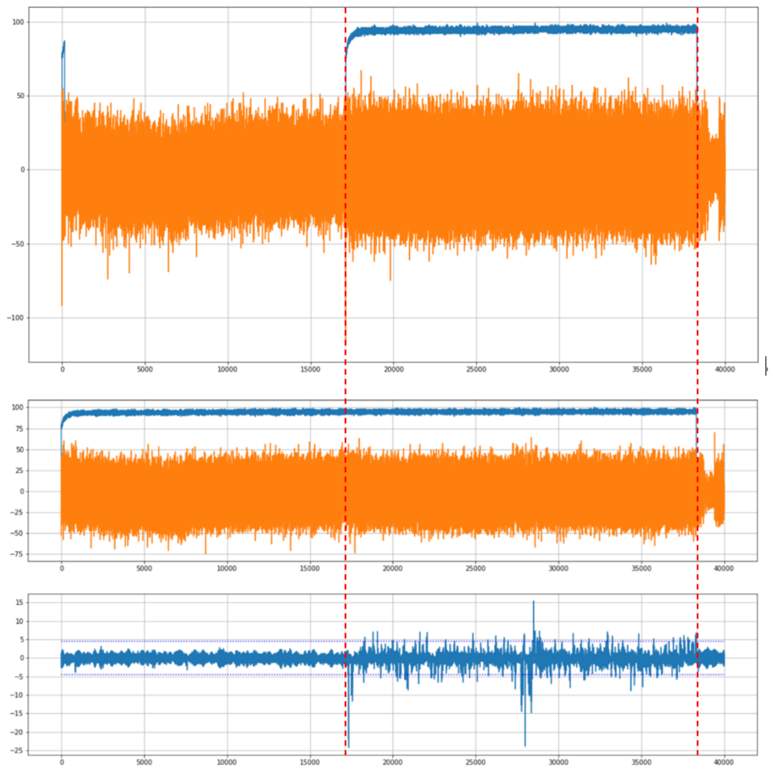


Figure 6: The part inside the red line is the computation of the plaintext and ad (encryption with the fixed message vs. random message)

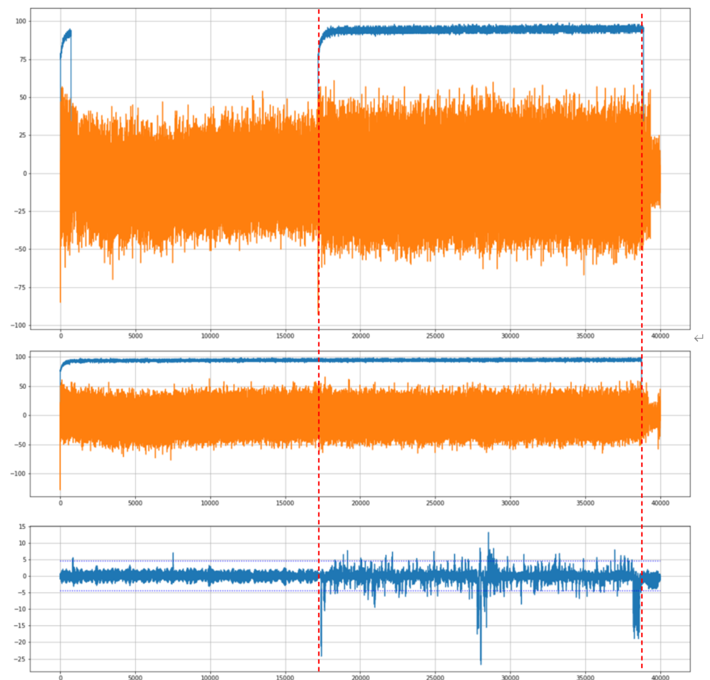


Figure 7： The part inside the red line is the computation of the ciphertext and ad (decryption with the fixed message vs. random message)

For the fixed key vs. random key t-test, the keyschedule function is marked witha red dotted line in Figures 8 and 9 below. It can be seen there are leaks in the encryption and decryption, except for keyschedule part.

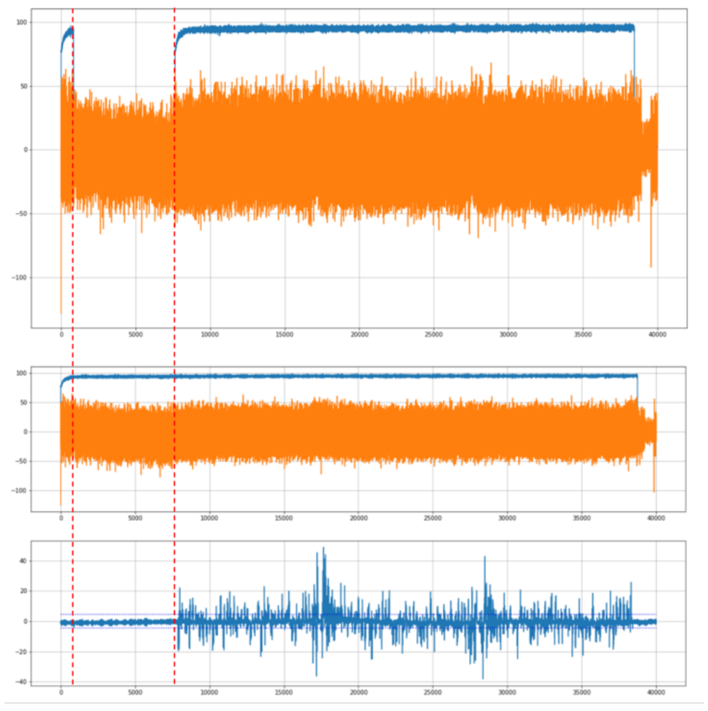


Figure 8: The part inside the red line is the keyschedule function (encryption with the fixed key vs. random key)

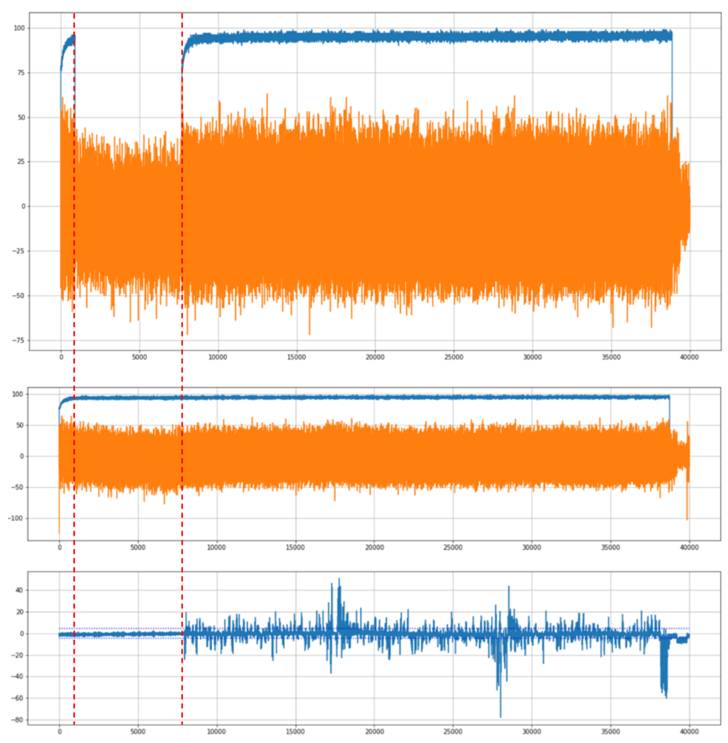


Figure 9: The part inside the red line is the keyschedule function (decryption with the fixed key vs. random key)

For our input test vector (shown in Table 1), the giftb128_encrypt_block function needs to be run three times. Through trigger signal localization, we found that two large leakage spikes appeared between these three giftb128_encrypt_blocks, as shown in Figure 10.

Table 1: Input/output details of GIFT-COFB

Fixed input/output	**Value**
Key	000102030405060708090A0B0C0D0E0F
Nonce	000102030405060708090A0B0C0D0E0F
PT	00010203
AD	00010203
CT	ACA0E4DAA2F7C53C377BD7B30FFC434CD892F909

To find the cause of these two leakage spikes, we check the ASM code inside giftb128_encrypt_blocks and find that the internal state is unmasked at the end of the function. This removes the randomness introduced by the round key, resulting in the intermediate state of the giftb128_encrypt_blocks output being a fixed value for the fixed input. So between the two giftb128_encrypt_blocks, the linear operation of the intermediate state shows a large leak.

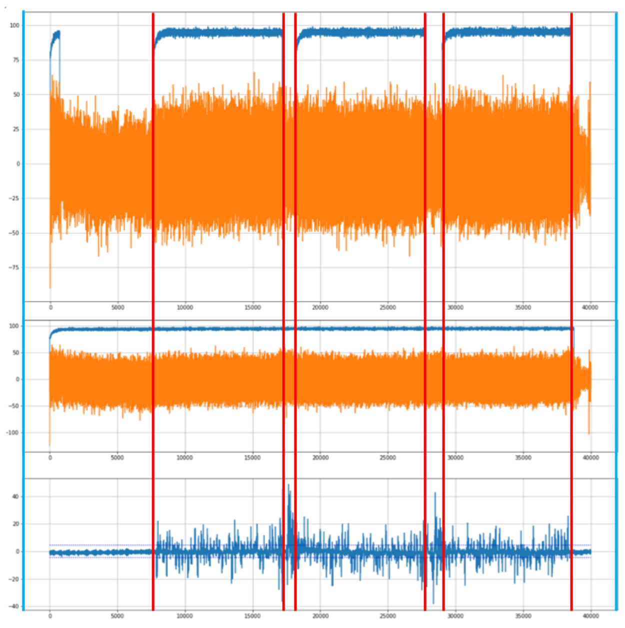


Figure 10: The three giftb128_encrypt_blocks is marked with a red line (encryption with the fixed key vs. random key)

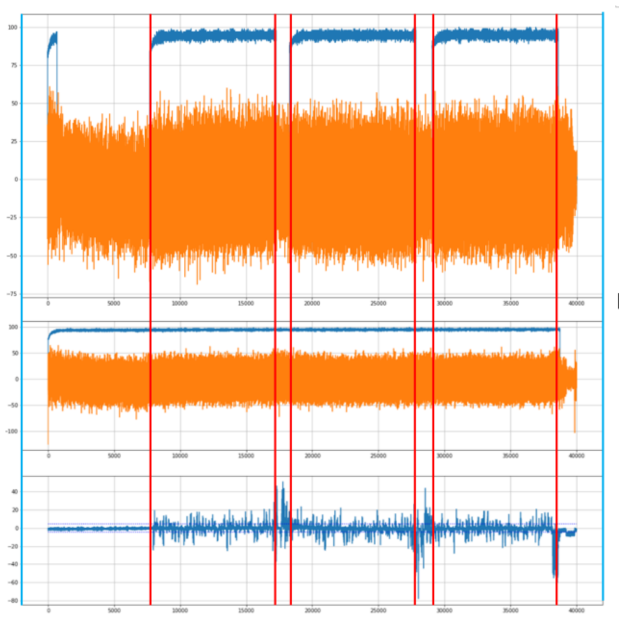


Figure 11: The three giftb128_encrypt_blocks is marked with a red line (decryption with the fixed key vs. random key)

For decryption, an extra spike is shown at the end of the last giftb128_encrypt_blocks in Figure 11, which is due to verifying that the calculated tag matches the fixed unshared input tag. At the same time, we also noticed that there is a leak inside giftb128_encrypt_blocks, which requires ad-hoc analysis.

References

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

[M20] Beyond Birthday Bound Secure Fresh Rekeying: Application to Authenticated Encryption. ASIACRYPT (1) 2020: 630-661

[PSV15] Olivier Pereira, François-Xavier Standaert, Srinivas Vivek: Leakage-Resilient Authentication and Encryption from Symmetric Cryptographic Primitives. CCS 2015: 96-108