SCIENCE CHINA
Information Sciences

• **Supplementary File** •

Unbalanced sharing: a threshold implementation of SM4

Man WEI[1,2,3], Siwei SUN[1,2*], Zihao WEI[1,2,3] & Lei HU[1,2]

[1]*State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing* 100093*, China;*
[2]*Data Assurance and Communication Security Research Center, Chinese Academy of Sciences, Beijing* 100093*, China;*
[3]*School of Cyber Security, University of Chinese Academy of Sciences, Beijing* 100049*, China*

Appendix A Hardware architectures

In the following, we introduce encryption process and key expansion in detail. As can be seen in Fig. A1 (left), the register arrays X_0, X_1, X_2, X_3, each consists of four 16-bit (two 8-bit shares) width shift registers, and register array X_4 consists of three 24-bit (three 8-bit shares) width shift registers.

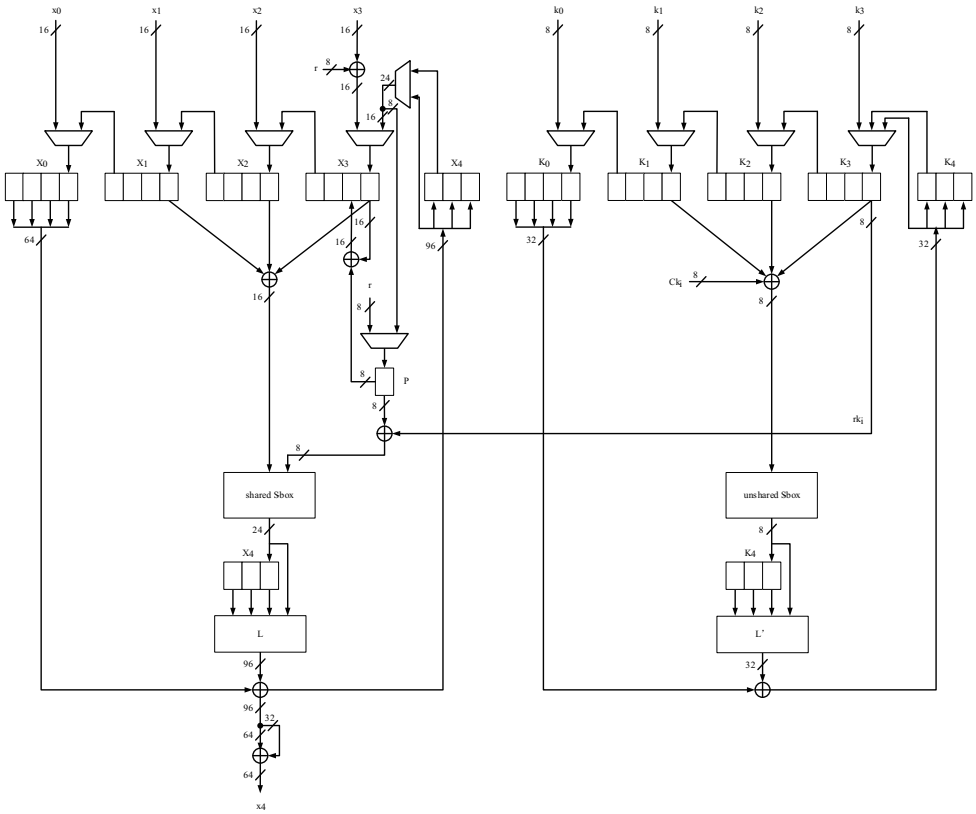

Figure A1 Hardware architecture of the masked SM4 encryption core

Our implementation requires 6 clock cycles to complete one round. In the first 4 clock cycles, data in the rightmost registers of X_0, X_1, X_2, X_3 updates, and is selected to perform the XOR operations with the corresponding round key. In

* Corresponding author (email: sunsiwei@is.ac.cn)

the first 4 cycles of first encryption round, 8-bit randomness is injected as the third share to shared S-box. Besides, the randomness is XORed with one input share to keep the correct mask. According to the discussion of TI-based SM4 S-box, 3 clock cycles are required to complete the S-box computation, and therefore the correct three output shares are available after a delay of 2 clock cycles. The three output shares of the shared S-box are stored in register arrays X_4 during the 4th clock cycles. After the shared S-box is operated 4 times, the forth output and the former three outputs in register arrays X_4 form three 32-bit shares, and each share performs the L linear transformation. Then the first two 32-bit output shares perform the XOR operations with two 32-bit shares from X_0 respectively. The first two shares of leftmost byte enters in the rightmost register of X_3 directly, while the third share enters the P register prepare to enter the shared S-box. Other bytes are stored in the register arrays X_4. When data in the rightmost register of X_3 is shifted to the left, the second share is XORed with the data in P to return two shares. Similarly, the ciphertext outputs with two shares by adding the third share to the second share. In Fig. A1, we describe X_4 in two places in order to clarify the different functions of X_4.

From the above, $6 \times 32 = 192$ cycles are required to complete one SM4 encryption. Key expansion has the same timing flow. The 128-bit input key MK has been initialized by XORed with 128-bit system parameter FK, and enters the registers K_0, K_1, K_2, K_3. To get the round key ready for the encryption round, key expansion is kept one round (6 clock cycles) ahead of encryption. The constant keys CK_i is generated dynamically. Every byte of CK_i is produced from the previous byte added with $0x7$ modulo 256 from 2nd clock cycle to 5th clock cycle, where the modulo operation can be implemented directly by keeping the low 8 bits. Thus, our complete implementations needed 198 clock cycles totally.

Appendix B Synthesis results

We synthesized the architecture of the serialized SM4 TI encryption core with Synopsys Design Compiler 2014.09-SP3 and UMC 180nm Standard cell library. We apply the *compile* and *compile_ultra* commands to each component of the SM4 implementation and the results are summarized in Table B1. Our implementation costs 7316 GE, 8-bit fresh randomness per S-box, and requires 198 clock cycles for one encryption. Besides, as we can see from Table B2, our shared SM4 S-box with unbalanced sharing is very competitive, which only takes 2000 GE, 8-bit fresh randomness, and 3 clock cycles.

Table B1 Implementation result for the SM4 TI encryption core

Area	[GE] *compile* / *_ultra*
shared S-box	2060 / 2000
Register arrays [1)]	3378 / 3134
Key Expansion	1948 / 1835
Control	358 / 347
Total	7744 / 7316
	[bits/per S-box]
Randomness	8
	[clock cycles]
Latency	198

Table B2 Implementation results for first-order secure Sbox designs of AES and SM4

S-box	Area[GE] *compile* / *_ultra*	Randomness [bits]	Clock Cycles
AES			
Moradi *et al.* [1]	no data / 4244	44	4
Bilgin *et al.* [2]	2835 / 2224	32	3
Cnudde *et al.* [3]	1977 / 1872	54	6
Ueno *et al.* [4]	1425[2)] / 1342[2)]	64	5
Gross *et al.* [5]	2200[3)]	20	8
Wegener *et al.* [6]	4200[3)]	0	16
SM4			
This work	2060 / 2000	8	3

1) not including the input and output linear transformation
2) not including the input and output linear transformation
3) without *compile_ultra* flag

Appendix C Leakage Analysis

We collect 1 million power traces when the PPRNG is off, and 10 million power traces when the PPRNG is on. Example power traces are shown in Fig. C1. For the first-order t-test, the results are shown in Fig. C2. As is commonly believed, the absolute t-value of less than 4.5 indicates a high confidence in the resistance against standard first-order attacks. When the PRNG is off, some leaks surpass the confidence threshold of ± 4.5. Thus, the implementation suffers the first-order leakages only under 10 million power traces. When the PRNG is on, the absolute t-value is below 4.5. Therefore, we conclude that our TI of SM4 is secure against standard first-order attacks under the condition of 10 million traces. On the other hand, as shown in Fig. C3, our TI exhibits obvious leakage with respect to second-order analysis.

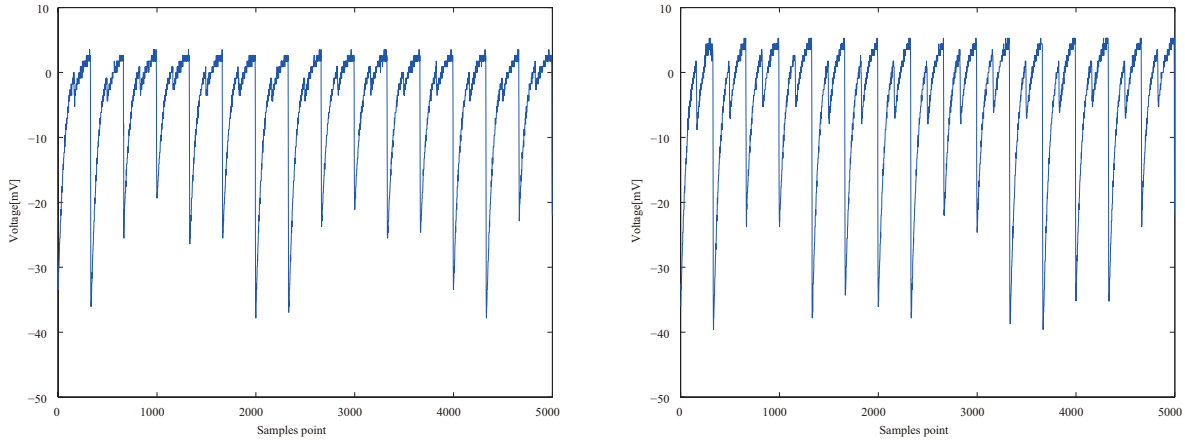

Figure C1 Power trace of the SM4 TI without PRNG (left) and with PRNG (right)

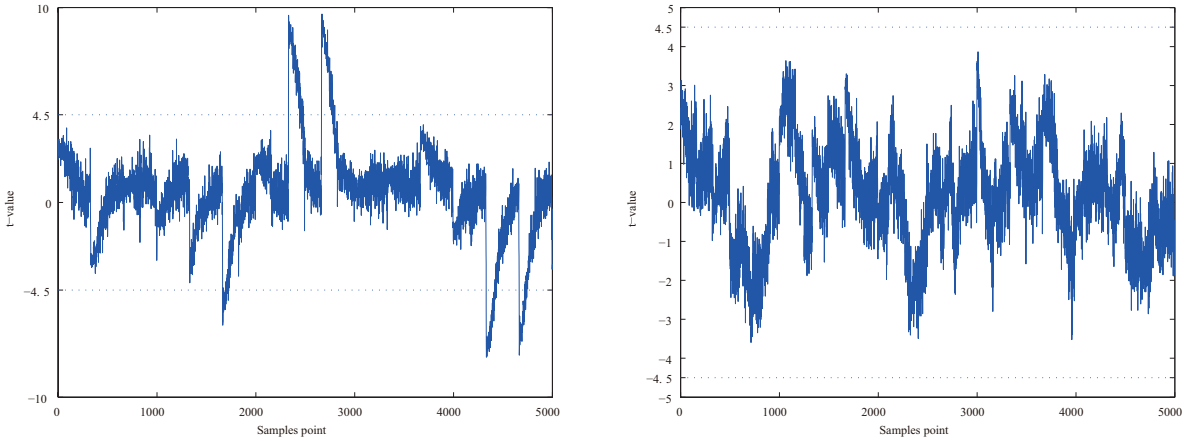

Figure C2 First-order t-test results without PRNG (left) and with PRNG (right)

References

1 Moradi A, Poschmann A, Ling S, et al. Pushing the limits: a very compact and a threshold implementation of AES. In: Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, 2011. 69–88

2 Bilgin B, Gierlichs B, Nikova S, et al. Trade-offs for threshold implementations illustrated on AES. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2015, 34(7): 1188–1200

3 De Cnudde T, Reparaz O, Bilgin B, et al. Masking AES with $d+1$ Shares in Hardware. In: Proceedings of International Conference on Cryptographic Hardware and Embedded Systems, Santa Barbara, CA, USA, 2016. 194–212

4 Ueno R, Homma N, Aoki T. Toward more efficient DPA-resistant AES hardware architecture based on threshold implementation. In: Proceedings of International Workshop on Constructive Side-Channel Analysis and Secure Design, Paris, France, 2017. 50–64

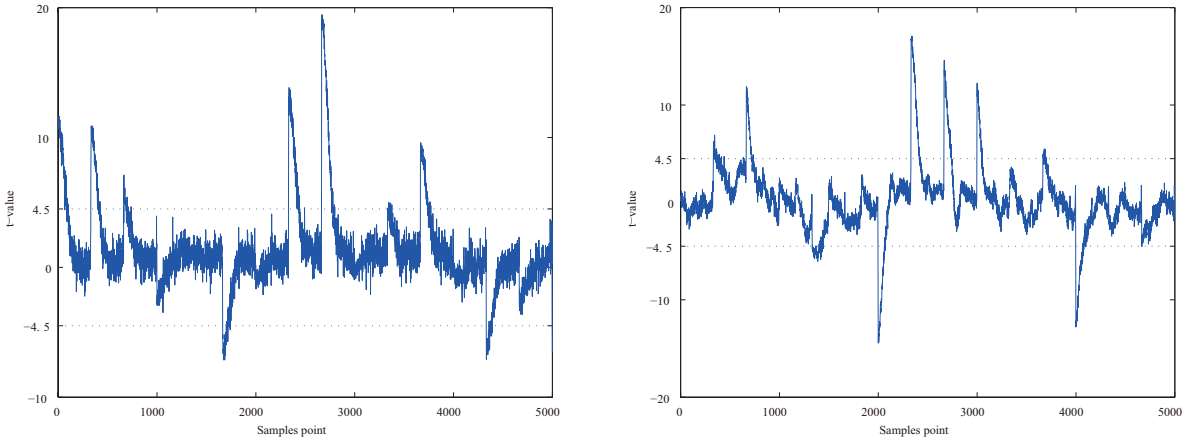

Figure C3 Second-order t-test results without PRNG (left) and with PRNG (right)

5 Gross H, Mangard S, Korak T. An efficient side-channel protected AES implementation with arbitrary protection order. In: Proceedings of Cryptographers? Track at the RSA Conference, San Francisco, CA, USA, 2017. 95–112

6 Wegener F, Moradi A. A First-Order SCA Resistant AES Without Fresh Randomness. In: Proceedings of International Workshop on Constructive Side-Channel Analysis and Secure Design, Singapore, 2018. 245–262