Hindawi
Security and Communication Networks
Volume 2023, Article ID 8044164, 16 pages
https://doi.org/10.1155/2023/8044164

Research Article

Efficient (Masked) Hardware Implementation of Grain-128AEADv2

Bohan Li,[1,2] **Hailong Zhang**,[1,2] **and Dongdai Lin**[1,2]

[1]*State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China*
[2]*School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China*

Correspondence should be addressed to Hailong Zhang; zhanghailong@iie.ac.cn

Received 10 November 2022; Revised 31 January 2023; Accepted 16 February 2023; Published 21 April 2023

Academic Editor: Jie Cui

We consider the efficient hardware implementation of Grain-128AEADv2, which is the second version of Grain-128AEAD (one of the lightweight cryptography finalist candidates). In order to counteract side-channel attacks, the efficient masked hardware implementation of Grain-128AEADv2 is also considered under the idea of domain-oriented masking. In detail, the so-called pipeline-like pre-computation technique is applied to increase the throughput-area ratio of the (masked) hardware implementation of Grain-128AEADv2. The performance of the (masked) hardware implementation of Grain-128AEADv2 is evaluated on ASIC and FPGA. For the unmasked version, the highest throughput-area ratio can be 2.14 Mbps/*GE* on ASIC and 9.34 Mbps/Slice on FPGA. For the masked version, the highest throughput-area ratio can be 0.37 Mbps/*GE* on ASIC and 1.72 Mbps/Slice on FPGA. Then, the security of the masked hardware implementation of Grain-128AEADv2 is verified with the simulated *T*-Test. To the best of our knowledge, this is the first published work about the (masked) hardware implementation of Grain-128AEADv2. In light of this, this contribution may help researchers and practitioners to accurately compare the efficiency and the security of the hardware implementation of Grain-128AEADv2 with those of other lightweight cryptography algorithms.

1. Introduction

In August 2018, National Institute of Standards and Technology (NIST) launched the project Lightweight Cryptography (LWC) calling for lightweight cryptographic algorithms that possess the function of authenticated encryption with associated data (AEAD). These algorithms are expected to be suitable to be used in resource-constrained environments where existing cryptographic algorithms may be unsuitable to be used. Compared with existing symmetric cryptographic algorithms, lightweight cryptographic algorithms that possess the function of AEAD can achieve the goal of integrity and confidentiality simultaneously. Then, after two rounds of review, NIST announced in March 2021 that Grain-128AEAD and nine other lightweight cryptographic algorithms are selected as the finalists.

The Grain family was first proposed as a candidate of the eSTREAM project [1] launched by the European Network of Excellence for Cryptology. After three rounds of evaluation, Grain v1 [2] of the Grain family was chosen as one of the three stream ciphers included in Portfolio 2 (hardware-oriented). Then, recognizing the emerging need for 128-bit keys, Hell et al. proposed Grain-128 [3], which can support 128-bit keys and 96-bit IVs. Then, Grain-128a [4] that supports authentication was proposed in 2011. Based on Grain-128a, Grain-128AEAD [5] was proposed and submitted to the project LWC. Compared with Grain-128a, Grain-128AEAD can additionally use associated data to support authentication. Finally, in order to add security against key reconstruction from a known internal state, Grain-128AEADv2 [6] was proposed and submitted to the project LWC as a finalist.

Considering that lightweight cryptographic algorithms are expected to be used in resource-constrained environment, it is very meaningful to evaluate the performance of the hardware implementation of Grain-128AEADv2. In fact, there have been several hardware implementations of the Grain family. In [7], the hardware implementation of

Grain-128a was proposed. In [8], the efficient hardware implementation of Grain-128AEAD was proposed. In order to increase the throughput-area ratio, the Galois transformation of the NFSR was adopted in these hardware implementations. However, compared with the original ones, such transformation would lead to different output sequences and different states of NFSR. Moreover, since different versions of the hardware implementation of Grain-128AEAD adopt different Galois feedback functions, the output sequences of different versions of the hardware implementation of Grain-128AEAD can be also different, which will decrease the generality of the hardware implementation of Grain-128AEAD. In order to generate the same output sequences with the original ones, the secret key should be transformed [9]. The transformation of the secret key needs extra control logic. The situation can be more complicated in the hardware implementation of Grain-128AEADv2 since the secret key bits are XORed with the input to both LFSR and NFSR in the key reintroducing phase. Thus, extra memory can be needed to save the transformed secret key, which will increase the area cost of the hardware implementation of Grain-128AEADv2. Apart from that, the increased area cost and the more complicated control logic can influence the synthesis process and decrease the throughput-area ratio of the hardware implementation of Grain-128AEADv2.

One strategy to increase the throughput-area ratio of the hardware implementation of Grain-128AEADv2 is to apply the pipeline technique, which aims to increase the frequency of the hardware implementation of Grain-128AEADv2 by inserting registers into the critical path. However, it is not possible to apply the pipeline technique for FSRs due to their intrinsic feedback property [8]. In light of this, a pipeline--like pre-computation technique is proposed. Compared with the Galois transformation, the pipeline-like pre-computation technique has two advantages. First, it does not change the feedback functions, which can lead to the same output sequence as the original Grain-128AEADv2. Thus, it will not influence the generality of the hardware implementation of Grain-128AEADv2. Second, since the feedback functions are not changed, the parallel level of the hardware implementation of Grain-128AEADv2 with the pipeline-like pre-computation technique can be up to x32, which can be two times the maximum parallel level (x16) reached by the hardware implementation of Grain-128AEADv2 with the Galois transformation [7]. As the throughput-area ratio of a hardware implementation increases with its parallel level, it can achieve an efficient hardware implementation of Grain-128AEADv2.

In practice, side-channel attacks can exploit the leakage of a cryptographic algorithm implementation to recover the secret key, which can pose a serious threat on the security of a cryptographic algorithm implementation. In light of this, the hardware implementation of Grain-128AEADv2 that is secure against side-channel attacks should be considered. In fact, there exist different types of countermeasures that can resist side-channel attacks, such as masking [10, 11], shuffling [12], and random delay [13]. Among them, masking as a provably secure countermeasure can be the most famous one. Therefore, the masking technique should be adopted in the secure hardware implementation of Grain-128AEADv2. Since the idea of masking was proposed at CRYPTO 1999 [14], masking schemes suitable to be used for software and hardware implementations have been proposed over the past twenty years. Compared with the software ones, the hardware ones may face the security problems related to glitch. The security of a masking scheme should be analyzed in a certain adversary model. Ishai et al. for the first time proposed the d-probing model at CRYPTO 2003 [11], where the adversary can obtain the values of d bits with d probes. However, the security of the hardware implementation of a cryptographic algorithm under the d-probing model may be not enough since the leakage related to *glitch* is not considered [15]. For example, the authors of [16, 17] applied side-channel attacks on masked AES hardware implementations, and they led to the conclusion that glitch can pose a serious threat on the security of masked AES hardware implementations.

In order to consider the security problems related to glitch, the idea of the glitch-extended probing model was first proposed at CRYPTO 2015 [18]. Then, its formal version was proposed at CHES 2018 [19]. The first glitch-resistant masking scheme, *i.e.*, *Threshold Implementation (TI)*, was proposed by Nikova et al. at ICISC 2006. At least $td + 1$ shares should be used in TI where t denotes the degree of a non-linear function and d denotes the security order. Therefore, the number of shares needed in TI can be large. Then, in order to decrease the number of shares needed in TI, Reparaz et al. proposed Consolidating Masking Schemes (CMSs) at CRYPTO 2015 [18] by using fresh randomness. After that, *Domain-Oriented Masking (DOM)* [20] and *Unified Masking Approach (UMA)* [21] were proposed to further reduce the number of fresh randomness. Among these schemes, DOM can induce the least computation delay and the minimum number of extra operations. Moreover, the number of fresh randomness required by DOM can be relatively small. Therefore, DOM may be suitable to be used to secure the efficient hardware implementation of Grain-128AEADv2. In order to increase the throughput-area ratio of the masked hardware implementation of Grain-128AEADv2, the pipeline-like pre-computation technique can be also applied. However, for the parallel level above 16, the masked feedback functions will use some values that have not yet been shifted into the FSRs, and pre-computation cannot be processed. Thus, only the parallel level up to 8 can be achieved in the masked hardware implementation of Grain-128AEADv2. In fact, the security of the masked hardware implementation of Grain-128AEADv2 is verified with T-Test proposed by Gilbert Goodwill et al. [22] in simulated scenario.

Then, the performance of the (masked) hardware implementation of Grain-128AEADv2 is evaluated on both ASIC and FPGA. According to the synthesis results, the hardware implementation of Grain-128AEADv2 with the pipeline-like pre-computation technique can obtain the highest throughput-area ratio for both unmasked version and masked version. In detail, the highest throughput-area ratio for the unmasked version can be obtained in the x32

parallel version with the pipeline-like pre-computation technique, which is 2.14 Mbps/*GE* on ASIC and 9.34 Mbps/Slice on FPGA; the highest throughput-area ratio for the masked version can be obtained in the x8 parallel version with the pipeline-like pre-computation technique, which is 0.37 Mbps/*GE* on ASIC and 1.72 Mbps/Slice on FPGA. Overall, this contribution may help researchers and practitioners to accurately compare the hardware implementation efficiency of Grain-128AEADv2 with those of other lightweight cryptography algorithms.

The rest of the paper is organized as follows. Preliminaries are presented in Section 2. Then, three versions of hardware implementation of Grain-128AEADv2 are shown in Section 3. In Section 4, the masked hardware implementation of Grain-128AEADv2 with the pipeline-like pre-computation technique is shown. Then, in Section 5, the performance of the (masked) hardware implementation of Grain-128AEADv2 is evaluated on both ASIC and FPGA. In Section 6, *T*-Test is used to evaluate the security of the masked hardware implementation of Grain-128AEADv2 in the simulated scenario. Finally, conclusions are drawn in Section 7.

2. Preliminaries

First, the details of Grain-128AEADv2 are presented; second, the glitch-extended probing model is presented; third, the DOM masking scheme is presented.

2.1. Grain-128AEADv2. Grain-128AEADv2 is composed of two building blocks [6]. The first block is a pre-output generator, which is composed of a Linear Feedback Shift Register (LFSR), a Non-linear Feedback Shift Register (NFSR), and a pre-output function. The second block is an authenticator generator, which is composed of a shift register and an accumulator. The structure of Grain-128AEADv2 is shown in Figure 1.

The pre-output generator consists of a 128-bit LFSR S, a 128-bit NFSR B, and a pre-output function y. It generates a stream of pseudo-random bits, which can be used for encryption and authentication. The states of 128-bit LFSR and 128-bit NFSR at clock cycle t can be denoted as $S^t = [s_0^t, s_1^t, \cdots, s_{127}^t]$ and $B^t = [b_0^t, b_1^t, \cdots, b_{127}^t]$, respectively. The corresponding update functions of LFSR and NFSR can be expressed with $f(S_t)$ and $g(B_t)$:

$$\begin{aligned} s_{127}^{t+1} &= f(S_t) \\ &= s_0^t + s_7^t + s_{38}^t + s_{70}^t + s_{81}^t + s_{96}^t, \\ b_{127}^{t+1} &= s_0^t + g(B_t) \\ &= s_0^t + b_0^t + b_{26}^t + b_{56}^t + b_{91}^t + b_{96}^t + b_3^t b_{67}^t + b_{11}^t b_{13}^t \\ &\quad + b_{17}^t b_{18}^t + b_{27}^t b_{59}^t + b_{40}^t b_{48}^t + b_{61}^t b_{65}^t + b_{68}^t b_{84}^t \\ &\quad + b_{22}^t b_{24}^t b_{25}^t + b_{70}^t b_{78}^t b_{82}^t + b_{88}^t b_{92}^t b_{93}^t b_{95}^t. \end{aligned} \tag{1}$$

The output of the pre-output generator is given by the pre-output function y as

$$\begin{aligned} y_t &= y(S_t, B_t) \\ &= b_{12}^t s_8^t + s_{13}^t s_{20}^t + b_{95}^t s_{42}^t + s_{60}^t s_{79}^t + b_{12}^t b_{95}^t s_{94}^t \\ &\quad + s_{93}^t + b_2^t + b_{15}^t + b_{36}^t + b_{45}^t + b_{64}^t + b_{73}^t + b_{89}^t. \end{aligned} \tag{2}$$

The authenticator generator consists of a 64-bit shift register R and a 64-bit accumulator A. We denote the content of the shift register at instance i as $R_i = [r_0^i, r_1^i, \cdots, r_{63}^i]$ and the content of the accumulator at instance i as $A_i = [a_0^i, a_1^i, \cdots, a_{63}^i]$. The running of Grain-128AEADv2 consists of two phases: an initialization phase and a keystream generation phase. During the initialization phase, the LFSR and NFSR are first loaded with the key bits and IV bits. If we denote the key bits as $k_i, 0 \le i \le 127$ and the IV bits as $IV_i, 0 \le i \le 95$, all 128 bits of NFSR are loaded with the key bits, *i.e.*, $b_i^0 = k_i, 0 \le i \le 127$, and the first 96 LFSR bits are loaded with the IV bits, *i.e.*, $s_i^0 = IV_i, 0 \le i \le 95$. The last 32 bits of the LFSR are filled with 31 ones and a zero, *i.e.*, $s_i^0 = 1, 96 \le i \le 126, s_{127}^0 = 0$. Then, the Grain-128AEADv2 is clocked 320 times, feeding back the pre-output function y and XORing it with the input to both the LFSR and the NFSR, *i.e.*,

$$\begin{aligned} s_{127}^{t+1} &= f(S_t) + y_t, \quad 0 \le t \le 319, \\ b_{127}^{t+1} &= s_0^t + g(B_t) + y_t, \quad 0 \le t \le 319. \end{aligned} \tag{3}$$

Then, Grain-128AEADv2 is clocked 64 times, reintroducing the key and XORing it with the input to both the LFSR and the NFSR, *i.e.*,

$$\begin{aligned} s_{127}^{t+1} &= f(S_t) + y_t + k_{t-256}, \quad 320 \le t \le 383, \\ b_{127}^{t+1} &= s_0^t + g(B_t) + y_t + k_{t-320}, \quad 320 \le t \le 383. \end{aligned} \tag{4}$$

Once the pre-output generator has been initialized, the authenticator generator is initialized by loading the register and the accumulator with the pre-output keystream as

$$\begin{aligned} a_j^0 &= y_{384+j}, \quad 0 \le j \le 63, \\ r_j^0 &= y_{448+j}, \quad 0 \le j \le 63. \end{aligned} \tag{5}$$

While the register and the accumulator are initializing, the LFSR and the NFSR should be simultaneously updated as

$$\begin{aligned} s_{127}^{t+1} &= f(S_t), \quad 320 \le t \le 383, \\ b_{127}^{t+1} &= s_0^t + g(B_t), \quad 320 \le t \le 383. \end{aligned} \tag{6}$$

Thus, when the Grain-128AEADv2 has been fully initialized, the LFSR and the NFSR states can be denoted as S_{512} and B_{512}, respectively. Besides, the register and the accumulator can be denoted as R_0 and A_0, respectively. During the keystream generation phase, the pre-output is used to generate z_i for encryption and z_i' for authentication. Here, z_i and z_i' can be defined as

$$\begin{aligned} z_i &= y_{512+2i}, \\ z_i' &= y_{512+2i+1}. \end{aligned} \tag{7}$$

Then, the message can be encrypted as

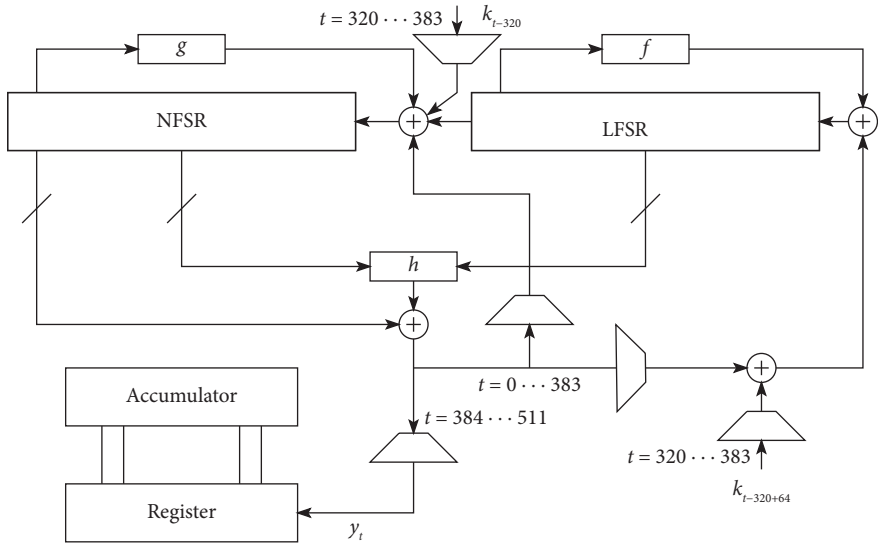

Figure 1: An overview of the building blocks in Grain-128AEADv2.

$$c_i = m_i \oplus z_i, \quad 0 \le i < L. \tag{8}$$

The accumulator A can be updated as

$$a_j^{i+1} = a_j^i + m_i r_j^i, \quad 0 \le j \le 63, 0 \le i \le L. \tag{9}$$

The shift register R can be updated as

$$\begin{aligned} r_{63}^{i+1} &= z_i', \\ r_j^{i+1} &= r_{j+1}^i, \quad 0 \le j \le 62. \end{aligned} \tag{10}$$

2.2. Glitch-Extended Probing Model. Usually, the security of masking schemes can be evaluated under the d–probing model [11], where the adversary can put up to d probes on intermediate variables of the implementation of a masking scheme. The d–probing model is formally defined in Definition 1.

Definition 1 (d–probing model [11]). Given a combinational logic circuit G, an adversary with d probes can observe up to d internal wires of G.

Glitch exists in the hardware implementation of a cryptographic algorithm. It means that by putting a probe on the output of G, one may obtain the input of G. Consequently, the security of hardware implementations of a cryptographic algorithm should be considered under the glitch-extended probing model. The glitch-extended probing model is formally defined in Definition 2.

Definition 2 (glitch-extended probing model [19]). Given a combinational logic circuit G, an adversary with glitch-extended probes can observe all the inputs of G up to the latest synchronization point by probing any output of G.

Example 1. Given a combinational logic circuit G which can implement function $g = G(x_1, \cdots, x_n)$, the inputs of G can be represented as x_i where $0 < i < n$ while the output of G can be represented by g. Under the glitch-extended probing model, one may obtain the inputs of G, i.e., x_i, by probing g.

2.3. Domain-Oriented Masking (DOM). Domain-Oriented Masking (DOM) can be used to secure the efficient hardware implementation of Grain-128AEADv2 under the glitch-extended probing model with $d(d+1)/2$ fresh randomness, where d denotes the security level. In DOM, the XOR and the AND masking gadgets should be used to replace the original XOR and AND. In order to achieve the first-order security, the origin-sensitive variables x and y should be divided into (A_x, B_x) and (A_y, B_y), respectively. The masking gadgets take (A_x, B_x) and (A_y, B_y) as input and return (A_q, B_q) as the output. Owing to the linear property of the XOR operation, the XOR masking gadget can be trivially achieved as

$$\begin{aligned} A_q &= A_x + A_y, \\ B_q &= B_x + B_y. \end{aligned} \tag{11}$$

However, the AND masking gadget can be more complicated, which is shown in Figure 2. The AND masking gadget performs three steps in order to map the input shares to the output shares, which can be referred to as calculation, resharing, and integration. In the calculation step, the actual multiplication is performed and the product terms A_xA_y, A_xB_y, B_xA_y, and B_xB_y can be obtained. In DOM, (A_xA_y, B_xB_y) are defined as the *inner-domain terms* and (A_xB_y, B_xA_y) are defined as the *cross-domain terms*. Then, in the resharing step, each cross-domain term should be randomized with a fresh random Z so that it is independent of other terms. Therefore, it can be added to any arbitrary domain in the next step. In order to prevent that any glitch propagates through the resharing step, a register must be inserted at the end of the resharing step. Finally, in the integration step, the inner-domain terms and the reshared cross-domain terms are added to obtain (A_q, B_q).

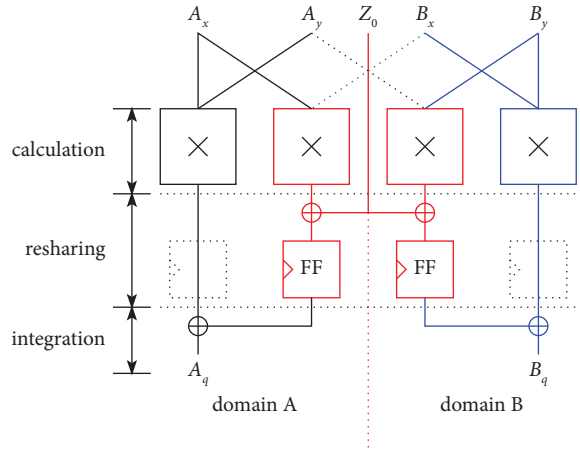

FIGURE 2: The 1st order DOM AND masking gadget.

3. Efficient Hardware Implementations of Grain-128AEADv2

In order to compare the hardware implementation performances of different LWC finalist candidates, the standard LWC hardware API [23, 24] is adopted in the hardware implementations of Grain-128AEADv2, which is presented in Section 3.1. Then, three versions of the hardware implementations of Grain-128AEADv2, *i.e.*, the straightforward version, the Galois transformation version, and the pipeline-like pre-computation version, are presented.

3.1. The Standard LWC Hardware API. The standard LWC hardware API includes the minimum compliance criteria, interface, communication protocol, and timing characteristics that should be supported by hardware implementation of Grain-128AEADv2. The interface of the hardware implementation of Grain-128AEADv2 is shown in Figure 3.

According to their different functions, the I/O ports can be divided into the *datapath ports* and the *control ports*. The datapath ports consist of the *data input ports* and the *data output ports*. The data should be input through ports *key* and *bdi_data* (block data input), and the data can be output through port *bdo_data* (block data output). The *key* port is controlled by the handshake signals *key_valid* and *key_ready*. *key_update* is used to notify that the internal key should be updated. The *bdi_data* port is controlled by the handshake signals *bdi_valid* and *bdi_ready*. The *bdi_valid_bytes* port and the *bdi_size* port indicate the location and the size of the valid data in the *bdi_data* port. The *bdo_data* port is controlled by the handshake signals *bdo_valid* and *bdo_ready*. Each width of *key*, *bdi_data* and *bdo_data* is set to 32 bits, the widths of *bdi_valid_bytes* and *bdi_size* are set to 4 and 3 bits, respectively, and the widths of other control ports are set to 1 bit so that they are consistent with the LWC hardware API.

3.2. The Straightforward Version. In order to analyze the structural characteristic and obtain the basic hardware implementation performance of Grain-128AEADv2, the straightforward version is implemented. The straightforward version of the hardware implementations of Grain-128AEADv2 follows the architectural design in [6]. The FSRs are in Fibonacci configuration, and the update functions are the same as the ones shown in Section 2. The FSRs are normally defined to be able to update one bit at each clock cycle. However, the design of the update functions makes it possible to calculate up to 32 update bits that can be used to update the FSRs in 32 continuous rounds in parallel. The throughput-area ratio increases with the parallel level of a hardware implementation. In light of this, the parallel versions up to x32 are implemented to optimize the throughput-area ratio of the hardware implementation of Grain-128AEADv2. For a given parallel level p, the highest throughput-area ratio of a hardware implementation can be related to the highest frequency that can be achieved, while the highest frequency of a hardware implementation depends on the *critical path* which corresponds to the maximal delay of two flip-flops. Similar to [8], the potential critical paths of Grain-128AEADv2 should be in the following ones:

(i) D_n: the maximal delay from any NFSR or LFSR flip-flop to any other NFSR or LFSR flip-flop.

(ii) D_y: the maximal delay from any NFSR or LFSR flip-flop to the output via the function y.

(iii) D_{ya}: the maximal delay from any NFSR or LFSR flip-flop to any accumulator flip-flop via the function y.

(iv) D_a: the maximal delay from any flip-flop in the authentication section to any accumulator flip-flop or output.

(v) D_{yn}: the maximal delay from any flip-flop of the NFSR or LFSR to any flip-flop of the NFSR via the function y.

Although there are several potential critical paths, some can be excluded by analyzing the update functions of Grain-128AEADv2. The update functions of the NFSR and the LFSR in the initialization phase can be more complicated than those in the generation phase because y needs to be additionally XORed to f and g. Then, D_n and D_y can be

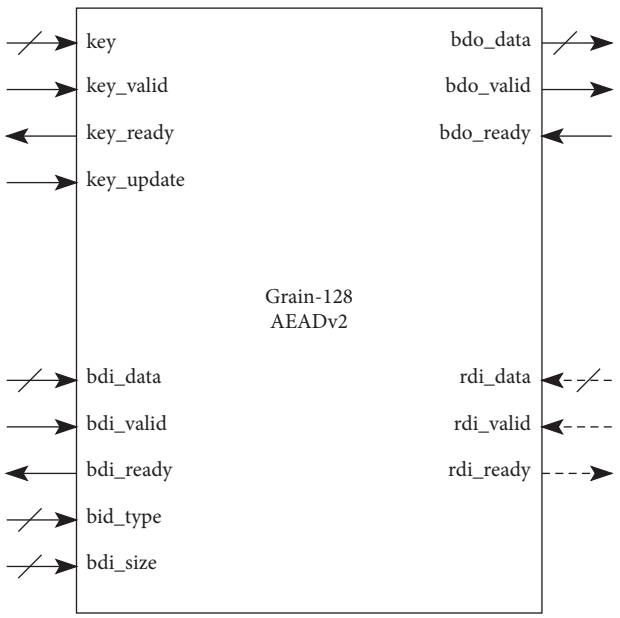

Figure 3: The interface of the hardware implementation of Grain-128AEADv2.

excluded because of the existence of D_{yn}. According to (3), (9), and (10), D_{yn} can be longer than D_{ya} and D_a. However, when we implement Grain-128AEADv2 in parallel, the data that are needed by the update function of the accumulator are not located yet. In this case, D_{ya} may be longer than D_{yn}, and it may be the critical path. The straightforward version of Grain-128AEADv2 is synthesized on both ASIC and FPGA, and the synthesis results on both the ASIC and FPGA can be seen in Table 1.

According to Table 1, the critical paths of hardware implementations of Grain-128AEAD and Grain-128AEADv2 can be identical, *i.e.*, critical paths of x1, x2, x4, and x8 implementations are D_{yn} and the critical paths of x16 and x32 implementations are D_{ya}. Then, in order to increase the frequency of the hardware implementation of Grain-128AEADv2, some strategies will be applied to optimize D_{yn} and D_{ya}.

3.3. The Galois Transformation Version. In order to decrease D_{yn}, one should transform the Fibonacci configuration to a Galois configuration. Besides, one needs to make sure that the Fibonacci configuration and the Galois configuration are equivalent so that their sets of output sequences can be identical. According to Table 1, the critical paths of x1, x2, x4, and x8 versions of the hardware implementation of Grain-128AEADv2 can be D_{yn} and those of x16 and x32 versions of the hardware implementation of Grain-128AEADv2 can be D_{ya}. The state of the NFSR can be denoted as $(x_0, x_1, \cdots, x_{n-1})$. The value of x_i should be updated by g_i, $0 \le i \le n-1$. For $0 \le i \le n-2$, we have $g_i = x_{i+1}$. Note that g_{n-1} can be defined as a complicated feedback function, and it may become the critical path in the hardware implementation of Grain-128AEADv2. On the contrary, in the Galois configuration, the expression of g_i can depend on more than one element of the NFSR. Thus, g_i in the Galois configuration can be simpler than g_{n-1} in the Fibonacci configuration, and the delay of the update phase of the Galois configuration may be shorter than that of the update phase of the Fibonacci configuration, which may optimize the frequency of the hardware implementation of Grain-128AEADv2. Since g_{n-1} in the Fibonacci configuration of Grain-128AEADv2 can be the same as that of Grain-128a and that of Grain-128AEAD, the Galois transformation of Grain-128AEADv2 can be identical to that of Grain-128a and that of Grain-128AEAD. Details about the feedback functions after the Galois transformation of Grain-128a can be seen in [7]. According to [7], the Galois transformation of Grain-128AEADv2 cannot be applied when the parallel level of the hardware implementation of Grain-128AEADv2 is above 16.

3.4. The Pipeline-Like Pre-Computation Version. In order to shorten D_{yn}, we split g and y into two parts. Then, the update phase of round r can be divided into two stages: (1) compute the first parts of g and y and (2) compute the second parts of g and y, XOR them with the first parts of g and y, and update the NFSR with the XORed values. When the pipeline technique is applied, stage 2 of round r and stage 1 of round $r+1$ should be computed at clock cycle t. However, the result of stage 2 of round r is used to update the NFSR and stage 1 of round $r+1$ can only be computed after the NFSR is updated. Thus, it seems that the pipeline technique cannot be applied. Considering that only the $(n-1)$-th bit of the NFSR is updated with the result of stage 2 and the other bits of the NFSR are updated with the $(i+1)$-th bit, stage 1 of round $r+1$ can be *pre-computed* before the NFSR is updated at clock cycle t. The *pipeline-like pre-computation* technique can be implemented as follows.

Compared with the straightforward version and the Galois transformation version, the pipeline-like pre-computation technique needs one more clock cycle. We denote this one more clock cycle as $t = -1$. At clock cycle $t = -1$, stage 1 of round 0 can be computed as

$$\begin{aligned} p_g &= s_0^0 + b_0^0 + b_{26}^0 + b_{56}^0 + b_{91}^0 + b_{96}^0 + b_3^0 b_{67}^0 + b_{11}^0 b_{13}^0 + b_{17}^0 b_{18}^0 + b_{27}^0 b_{59}^0, \\ p_y &= b_{95}^0 s_{42}^0 + s_{60}^0 s_{79}^0 + b_{12}^0 b_{95}^0 s_{94}^0 + s_{93}^0 + b_2^0 + b_{15}^0 + b_{36}^0 + b_{45}^0 + b_{64}^0 + b_{73}^0 + b_{89}^0. \end{aligned} \tag{12}$$

Table 1: Clock periods (ns) and critical paths (CP.) of the straightforward version of Grain-128AEAD and Grain-128AEADv2.

Cipher		x1	x2	x4	x8	x16	x32	Plat.
Grain-128AEAD [8]	Period	0.49	0.61	0.64	0.69	0.77	0.84	ASIC
	CP.	D_{yn}	D_{yn}	D_{yn}	D_{yn}	D_{ya}	D_{ya}	
Grain-128AEADv2	Period	0.478	0.480	0.492	0.520	0.591	0.597	ASIC
		2.76	2.81	2.85	2.90	3.08	4.04	FPGA
	CP.	D_{yn}	D_{yn}	D_{yn}	D_{yn}	D_{ya}	D_{ya}	

Note that the FSRs are not updated at clock cycle $t = -1$. At clock cycle $t = 0$, stage 2 of round 0 can be computed as

$$\begin{aligned} b_{127}^{t+1} &= p_g + b_{40}^t b_{48}^t + b_{61}^t b_{65}^t + b_{68}^t b_{84}^t \\ &\quad + b_{22}^t b_{24}^t b_{25}^t + b_{70}^t b_{78}^t b_{82}^t + b_{88}^t b_{92}^t b_{93}^t b_{95}^t, \\ y_{127}^{t+1} &= p_y + b_{12}^t s_8^t + s_{13}^t s_{20}^t. \end{aligned} \tag{13}$$

At the same time, stage 1 of round 1 should be pre-computed as

$$\begin{aligned} p_g &= s_1^t + b_1^t + b_{27}^t + b_{57}^t + b_{92}^t + b_{97}^t + b_4^t b_{68}^t + b_{12}^t b_{14}^t + b_{18}^t b_{19}^t + b_{28}^t b_{60}^t, \\ p_y &= b_{96}^t s_{43}^t + s_{61}^t s_{80}^t + b_{13}^t b_{96}^t s_{95}^t + s_{94}^t + b_3^t + b_{16}^t + b_{37}^t + b_{46}^t + b_{65}^t + b_{74}^t + b_{90}^t. \end{aligned} \tag{14}$$

Note that the FSRs begin to shift at clock cycle $t = 0$. Then, at clock cycle $t \geq 1$, stage 2 of round r can be computed similarly as (13) since the NFSR has been updated. Then, the pre-computation of stage 1 of round $r + 1$ can be computed with (14).

The parallel version of the pipeline-like pre-computation technique can work similarly. For example, for parallel level $p \leq 16$, the first part of g at stage 1 can be computed at clock cycle $t \geq 0$ as

$$\begin{aligned} p_{[p-1:0]}^g &= s_{[2p-1:p]}^t + b_{[2p-1:p]}^t + b_{[2p+25:p+26]}^t + b_{[2p+55:p+56]}^t + b_{[2p+90:p+91]}^t + b_{[2p+95:p+96]}^t + b_{[2p+2:p+3]}^t b_{[2p+66:p+67]}^t \\ &\quad + b_{[2p+10:p+11]}^t b_{[2p+12:p+13]}^t + b_{[2p+16:p+17]}^t b_{[2p+17:p+18]}^t + b_{[2p+26:p+27]}^t b_{[2p+58:p+59]}^t. \end{aligned} \tag{15}$$

However, for the parallel level $p = 32$, such split needs to be modified since some indexes of b can exceed 127, which can induce the result that the values of the bits with indexes exceeding 127 cannot be obtained in the current clock cycle.

In order to apply the pipeline-like pre-computation technique, the split of g and y should be modified. All the terms with indexes exceeding 127 are left to the second part. Thus, the first part of g and y can be modified as

$$\begin{aligned} p_g &= s_{[2p-1:p]}^t + b_{[2p-1:p]}^t + b_{[2p+25:p+26]}^t + b_{[2p+55:p+56]}^t + b_{[2p+10:p+11]}^t b_{[2p+12:p+13]}^t + b_{[2p+16:p+17]}^t b_{[2p+17:p+18]}^t \\ &\quad + b_{[2p+26:p+27]}^t b_{[2p+58:p+59]}^t + b_{[2p+39:p+40]}^t b_{[2p+47:p+48]}^t + b_{[2p+21:p+22]}^t b_{[2p+23:p+24]}^t b_{[2p+24:p+25]}^t, \\ p_y &= b_{[2p+11:p+12]}^t s_{[2p+7:p+8]}^t + s_{[2p+12:p+13]}^t s_{[2p+19:p+20]}^t + b_{[2p+1:p+2]}^t \\ &\quad + b_{[2p+14:p+15]}^t + b_{[2p+35:p+36]}^t + b_{[2p+44:p+45]}^t + b_{[2p+63:p+64]}^t. \end{aligned} \tag{16}$$

Then, the second part of g and y should be modified as

$$\begin{aligned} b_{127}^{t+1} &= p_g + b_{[p+90:91]} + b_{[p+95:96]} + b_{[p+2:3]} b_{[p+66:67]} + b_{[p+60:61]} b_{[p+64:65]} + b_{[p+67:68]} b_{[p+83:84]} \\ &\quad + b_{[p+69:70]} b_{[p+77:78]} b_{[p+81:82]} + b_{[p+87:88]} b_{[p+91:92]} b_{[p+92:93]} b_{[p+94:95]}, \\ y_{127}^{t+1} &= p_y + b_{[p+94:95]} s_{[p+41:42]} + s_{[p+59:60]} s_{[p+78:79]} + b_{[p+11:12]} b_{[p+94:95]} s_{[p+93:94]} + s_{[p+92:93]} + b_{[p+72:73]} + b_{[p+88:89]}. \end{aligned} \tag{17}$$

The second part of y for the parallel level $p = 32$ can be more complicated than that of y for other parallel levels. However, because of the large parallel level ($p = 32$), the effect of the longer path of D_{ya} can be relatively small.

3.5. Pipelining the Accumulator. Since only one of the generated bits is used for authentication, the update of the accumulator A with (9) can hold for the parallel level $p \le 2$. However, for the parallel level $p \ge 4$, A can be updated as [8]

$$a_j^{i+1} = a_j^i + \sum_{k=0}^{p/2-1} m_{i+k} \cdot r_{j+k}^i, \quad p \ge 4. \tag{18}$$

In such cases, some values have not yet been shifted into R. For example, for the parallel level $p = 16$, a_{63} can be updated as

$$a_{63}^{i+1} = a_{63}^i + \left(m_i \cdot r_{63}^i\right) + \left(m_{i+1} \cdot r_{64}^i\right) + \cdots + \left(m_{i+7} \cdot r_{70}^{i+7}\right), \tag{19}$$

where $(r_{64}^i, \cdots, r_{70}^{i+7})$ is being computed with (10) and is not shifted into R. Then, the update of A needs to wait for the computation of y, which means that the delay of D_{ya} can be longer than the delay of D_y. This is verified by the results shown in Table 1.

In order to make D_{ya} shorter, Sönnerup et al. [8] inserted a pipeline step between y and A and isolated A. However, in the Galois transformation version and the pipeline-like pre-computation version, we insert the pipeline step in the update of A for the parallel level $p \ge 16$ as shown in Figure 4. The advantage of such technique is that it needs less extra control logic. Therefore, it can lead to a relatively high frequency since the complicated control logic can have a negative effect on the frequency of the hardware implementation of Grain-128AEADv2. Note that the pipeline step will add one clock cycle delay to the update of A.

4. Masked Hardware Implementation of Grain-128AEADv2

Under the idea of DOM, the 1st order masked hardware implementation of Grain-128AEADv2 under the glitch-extended probing model is proposed.

4.1. The Straightforward Version. In order to achieve the d^{th} order security under the glitch-extended probing model, each sensitive variable in f, g, and y should be split into $d + 1$ shares. Thus, in the 1st order masked hardware implementation of Grain-128AEADv2, the LFSR and the NFSR should be split into two shares, *i.e.*, (S, S') and (B, B'). Then, the elements in S' and B' can be denoted as s_i' and b_i'. Besides, the 1st order masked f, g, and y can be denoted as F, G, and Y. Then, F can be implemented by applying f to each share as

$$\begin{aligned} s_{127}^{t+1} &= f\left(S_t\right) \\ &= s_0^t + s_7^t + s_{38}^t + s_{70}^t + s_{81}^t + s_{96}^t, \\ s_{127}^{\prime t+1} &= f\left(S_t'\right) \\ &= s_0^{\prime t} + s_7^{\prime t} + s_{38}^{\prime t} + s_{70}^{\prime t} + s_{81}^{\prime t} + s_{96}^{\prime t}. \end{aligned} \tag{20}$$

The non-linear terms in G and Y should be implemented with the AND masking gadget. Then, G and Y can be implemented by XORing the output shares of the AND masking gadget with the shares of the linear terms. DOM needs to insert one register stage into each AND masking gadget. The term of g and y with the highest degree is $b_{88}^t b_{92}^t b_{93}^t b_{95}^t$. Since each AND masking gadget of DOM needs to be inserted one register stage, the 1st order masked hardware implementation of the term $b_{88}^t b_{92}^t b_{93}^t b_{95}^t$ needs two register stages. The computation of $b_{88}^t b_{92}^t b_{93}^t b_{95}^t$ is shown in Figure 5.

The unmasked straightforward version encrypts 1 bit of the message every two clock cycles, while the masked straightforward version encrypts 1 bit of the message every six clock cycles. Therefore, the throughput-area ratio of the masked straightforward version will be much lower than that of the unmasked straightforward version.

4.2. The Pipeline-Like Pre-Computation Version. In order to increase the throughput-area ratio, the AND masking gadget can be implemented with the pipeline-like pre-computation technique. Since f is a linear function, F can be computed in one clock cycle. When the pipeline-like pre-computation technique is applied, the AND masking gadget can be computed in one clock cycle. In order to compute the AND masking gadget in one clock cycle, G and Y should be divided into three stages. In order to explain the details of the 1st order masked implementation of the Grain-128AEADv2 with the pipeline-like pre-computation technique, we take the masked update function G as an example. The 1st order masked hardware implementation of G with the pipeline-like pre-computation technique can be seen in Figure 6. The 1st order masked hardware implementation of Y can be similar to that of G.

According to Figure 6, G should be divided into three stages. The first stage aims to compute terms with degree 2 such as $e = b_3 b_{67}$ and $f = b_1 b_{13}$ with the AND masking gadget. Note that there are two terms with degree 2, *i.e.*, $m = b_{88}^t b_{92}^t$ and $n = b_{93}^t b_{95}^t$, in the term $b_{88}^t b_{92}^t b_{93}^t b_{95}^t$. The computation of e, f, m, and n can be shown as

$$\begin{cases} e_0^0 = b_3 b_{67}, \\ e_1^0 = b_3' b_{67}', \\ e_0^1 = b_3 b_{67}' + r_0, \\ e_1^1 = b_3 b_{67}' + r_0, \end{cases} \quad \begin{cases} f_0^0 = b_{11} b_{13}, \\ f_1^0 = b_{11}' b_{13}', \\ f_0^1 = b_{11} b_{13}' + r_1, \\ f_1^1 = b_{11} b_{13}' + r_1, \end{cases}$$
$$\begin{cases} m_0^0 = b_{88} b_{92}, \\ m_1^0 = b_{88}' b_{92}', \\ m_0^1 = b_{88} b_{92}' + r_9, \\ m_1^1 = b_{88}' b_{92} + r_9, \end{cases} \quad \begin{cases} n_0^0 = b_{93} b_{95}, \\ n_1^0 = b_{93}' b_{95}', \\ n_0^1 = b_{93} b_{95}' + r_{10}, \\ n_1^1 = b_{93}' b_{95} + r_{10}. \end{cases} \tag{21}$$

The XOR masking gadget of the linear terms in G can be computed with (22). The XOR masking gadget of the linear terms in Y can be computed similarly.

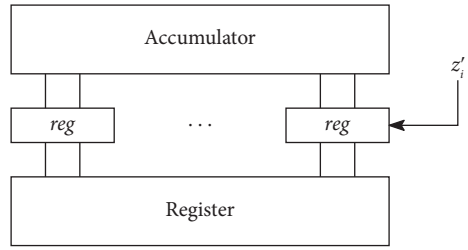

Figure 4: Pipelining the accumulator for the parallel level $p \geq 16$.

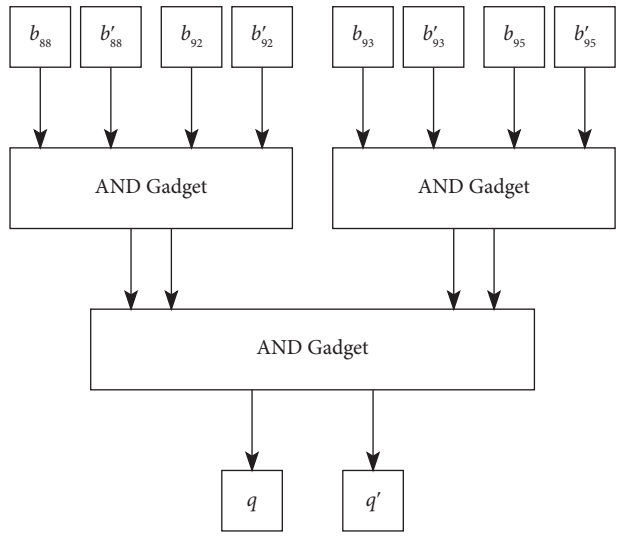

Figure 5: The 1st order masked hardware implementation of the term $b_{88}^t b_{92}^t b_{93}^t b_{95}^t$.

$$\begin{aligned} l_0 &= s_0 + b_0 + b_{26} + b_{56} + b_{91} + b_{96}, \\ l_1 &= s'_0 + b'_0 + b'_{26} + b'_{56} + b'_{91} + b'_{96}. \end{aligned} \tag{22}$$

The second stage uses an AND masking gadget to compute $q = mn$. Before computing $q = mn$, the shares of m and n should be XORed as $m_0 = m_0^0 + m_1^0$, $m_1 = m_0^1 + m_1^1$, $n_0 = n_0^0 + n_1^0$, and $n_1 = n_0^1 + n_1^1$ so that the effect of glitch does not exist any more. Then, the computation of $q = mn$ can be shown as

$$\begin{cases} q_0^0 = m_0 n_0, \\ q_1^0 = m_1 n_1, \end{cases} \begin{cases} q_0^1 = m_0 n_1 + r_{13}, \\ q_1^1 = m_1 b_0 + r_{13}. \end{cases} \tag{23}$$

At the same time, the obtained shares of terms with degree one and two can be XORed to reduce the computation complexity in the third stage, which can be shown as

$$\begin{aligned} p_0 &= l_0 + e_0 + f_0 + \cdots, \\ p_1 &= l_1 + e_1 + f_1 + \cdots. \end{aligned} \tag{24}$$

Then, in the third stage, G is computed and the two shares (B, B') of the NFSR can be updated as

$$\begin{aligned} b_{127} &= p_0 + \cdots + q_0, \\ b'_{127} &= p_1 + \cdots + q_1. \end{aligned} \tag{25}$$

The computation of G with the pipeline-like precomputation technique is shown in Figure 7. Compared with the straightforward version, the pipeline-like precomputation technique needs two more clock cycles. We denote these two more clock cycles as $t = -2$ and $t = -1$. Then, (b_{127}, b'_{127}) of round t can be obtained at every clock cycle $t \geq 0$. According to Figure 7, at clock cycle $t = -2$, only the stage 1 of round 0 is computed. At clock cycle $t = -1$, stage 2 of round 0 and the stage 1 of round 1 are computed. Since the shares of the FSRs are not shifted at $t = -2$, b_{i+1} and s_{i+1} should be used to compute stage 1 of round 1 at clock cycle $t = -1$. Then, at clock cycle $t \geq 0$, stage 3 of round t, stage 2 of round $t + 1$, and stage 1 of round $t + 2$ are computed. Since the shares of the FSRs are not shifted at $t = -1$, b_{i+1} and s_{i+1} should be used to compute stage 2 of round 1, and b_{i+2} and s_{i+2} should be used to compute stage 1 of round 2. Since the shares of the FSRs are shifted at clock cycle $t \geq 0$, the bits used to compute stage 1 and stage 2 can be the same.

Take the computation of the linear terms of G in stage 1 of round t for example. At clock cycle $t = -2$, the computation of the linear terms of G in stage 1 of round 0 should be pre-computed with (22). Since the shares of the FSRs are not shifted at $t = -2$ and $t = -1$, the computation of the linear terms of G in stage 1 of round 1 should be pre-computed at clock cycle $t = -1$ as

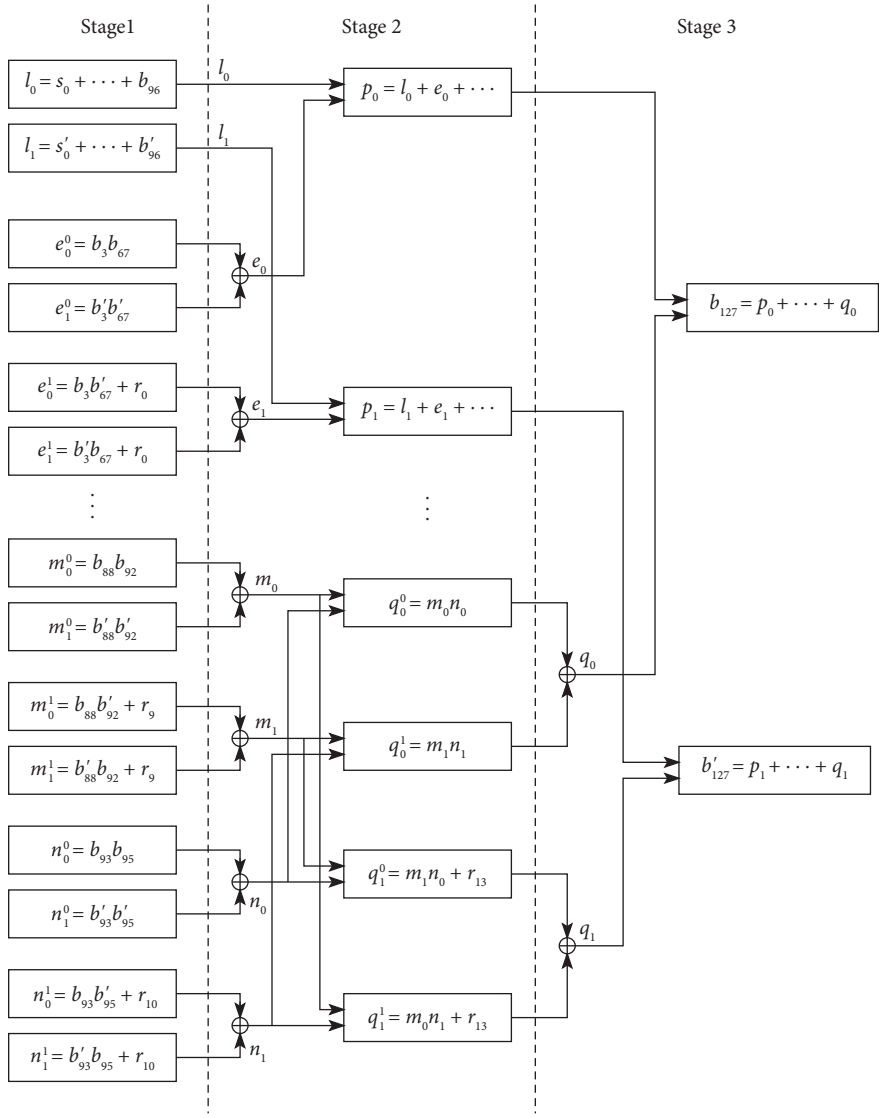

Figure 6: The 1st order masked hardware implementation of G.

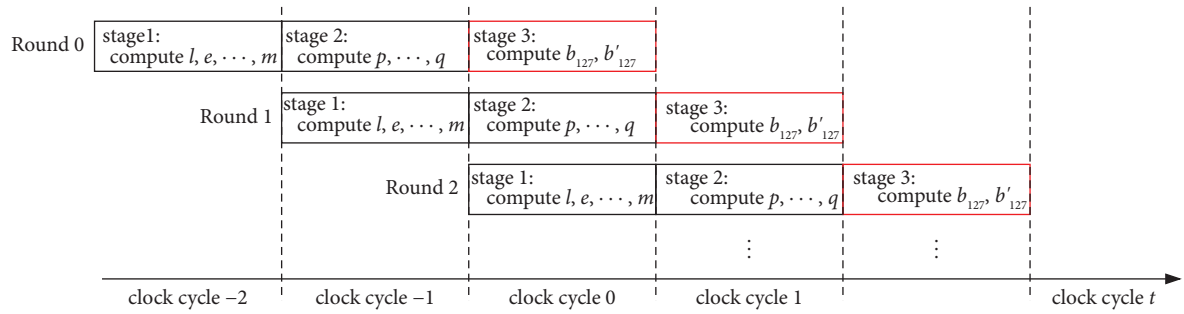

Figure 7: The 1st order masked hardware implementation of G with the pipeline-like pre-computation technique.

$$\begin{aligned} l_0 &= s_1 + b_1 + b_{27} + b_{57} + b_{92} + b_{97}, \\ l_1 &= s'_1 + b'_1 + b'_{27} + b'_{57} + b'_{92} + b'_{97}. \end{aligned} \tag{26}$$

Then, at clock cycle $t = 0$, the computation of the linear terms of G in stage 1 of round 2 should be pre-computed as

$$\begin{aligned} l_0 &= s_2 + b_2 + b_{28} + b_{58} + b_{93} + b_{98}, \\ l_1 &= s'_2 + b'_2 + b'_{28} + b'_{58} + b'_{93} + b'_{98}. \end{aligned} \tag{27}$$

Since the shares of the FSRs are updated at clock cycle $t \geq 0$, the computation of the linear terms of G in stage 1 of round $t + 2$ should be computed with (27).

In fact, the parallelization technique can be applied to the masked hardware implementation of the Grain-128AEADv2 with the pipeline-like pre-computation technique to increase its the throughput-area ratio. For example, at clock cycle t, the computation of G in stage 1 of round $t+2$ can be shown as

$$\begin{cases} e_0^0 = b_{[3p+2:2p+3]} b_{[3p+66:2p+67]}, \\ e_1^0 = b'_{[3p+2:2p+3]} b'_{[3p+66:2p+67]}, \\ e_0^1 = b_{[3p+2:2p+3]} b'_{[3p+66:2p+67]} + r_{[p-1:0]}, \\ e_1^1 = b_{[3p+2:2p+3]} b'_{[3p+66:2p+67]} + r_{[p-1:0]}, \end{cases}$$

$$\begin{cases} f_0^0 = b_{[3p+10:2p+11]} b_{[3p+12:2p+13]}, \\ f_1^0 = b'_{[3p+10:2p+11]} b'_{[3p+12:2p+13]}, \\ f_0^1 = b_{[3p+10:2p+11]} b'_{[3p+12:2p+13]} + r_{[2p-1p]}, \\ f_1^1 = b_{[3p+10:2p+11]} b'_{[3p+12:2p+13]} + r_{[2p-1p]}, \end{cases}$$

$$\vdots \tag{28}$$

$$\begin{cases} m_0^0 = b_{[3p+87:2p+88]} b_{[3p+91:2p+92]}, \\ m_1^0 = b'_{[3p+87:2p+88]} b'_{[3p+91:2p+92]}, \\ m_0^1 = b_{[3p+87:2p+88]} b'_{[3p+91:2p+92]} + r_{[11p-1:10p]}, \\ m_1^1 = b'_{[3p+87:2p+88]} b_{[3p+91:2p+92]} + r_{[11p-1:10p]}, \end{cases}$$

$$\begin{cases} n_0^0 = b_{[3p+92:2p+93]} b_{[3p+94:2p+95]}, \\ n_1^0 = b'_{[3p+92:2p+93]} b'_{[3p+94:2p+95]}, \\ n_0^1 = b_{[3p+92:2p+93]} b'_{[3p+94:2p+95]} + r_{[11p-1:10p]}, \\ n_1^1 = b'_{[3p+92:2p+93]} b_{[3p+94:2p+95]} + r_{[11p-1:10p]}. \end{cases}$$

When the pipeline-like pre-computation technique is applied, the parallel level can be up to x8. For the parallel level $p > 8$, some indexes of b can exceed 127, which can induce the result that the values of the bits with indexes exceeding 127 cannot be obtained in current clock cycle. Thus, such parallelization will lead to a longer critical path than the parallelization with $p \leq 8$, which may decrease the throughput-area ratio of the masked hardware implementation of Grain-128AEADv2. Consequently, only the parallel level with $p \leq 8$ is considered.

5. Performance Evaluation

In this section, the performance of the (masked) hardware implementation of Grain-128AEADv2 is evaluated on ASIC and FPGA. For the ASIC hardware platform, the STM 65 nm process with 1.2 V supply voltage and 25°C is adopted. The synthesis tool is the Synopsys Design Compiler L-2016.03-SP1. The FPGA hardware platform is the Xilinx Artix-7 family [25], and the *Synthesis* and the *Implementation* are conducted in Vivado 2020.1 [26] which is the standard IDE for Xilinx Artix-7 family with the Verilog hardware design language. Then, in order to precisely evaluate the performance of the (masked) hardware implementation of Grain-128AEADv2, only primary component *look up table (LUT)* in FPGA is used, and resources like SRL16/SRL32, BRAM, and DSP are not applied in our implementations. The fresh randomness is assumed to be generated by external Pseudo-Random Number Generator (PRNG). Therefore, the overhead of the generation of fresh randomness is not considered in the performance evaluation.

Overall, the evaluation results of the (masked) hardware implementation of Grain-128AEADv2 are shown in Tables 2 and 3.

According to Tables 2 and 3, the following three observations can be obtained.

(i) First, among different versions of the (masked) hardware implementation of Grain-128AEADv2, the (masked) hardware implementation of Grain-128AEADv2 with the pipeline-like pre-computation technique can reach the highest throughput-area ratio. For the unmasked version, the highest throughput-area ratio can be obtained with the parallel level $p = 32$, while for the masked version, the highest throughput-area ratio can be obtained with the parallel level $p = 8$. In detail, for the unmasked version, the highest throughput-area ratio of the hardware implementation of Grain-128AEADv2 can be 2.14 Mbps/*GE* on ASIC and 9.34 Mbps/Slice on FPGA, while for the masked version, that of the hardware implementation of Grain-128AEADv2 can be 0.37 Mbps/*GE* on ASIC and 1.72 Mbps/Slice on FPGA. Overall, compared to the other two versions, the increase rate of the throughput of the pipeline-like pre-computation version can be larger than the increase rate of the consumed area of the pipeline-like pre-computation version on both ASIC and FPGA. For example, for the unmasked version, the increase rate of the highest throughput of the pipeline-like pre-computation version to the straightforward version on ASIC can be 22.4%, while the increase rate of the consumed area of the pipeline-like pre-computation version to the straightforward version on ASIC can be only 13.9%. Therefore, the pipeline-like pre-computation version can obtain the highest throughput-area ratio on both ASIC and FPGA.

(ii) Second, the parallel level p can influence the throughput and the consumed area of the (masked) hardware implementation of Grain-128AEADv2. In fact, the throughput and the consumed area increase with the parallel level p. Since the increase rate of the throughput can be larger than that of the area, the throughput-area ratio of the (masked) hardware implementation of Grain-128AEADv2 can increase with the parallel level p. For example, for the unmasked version, the throughput of the straightforward version on ASIC can increase from 1.14 Gbps to 26.67 Gbps as the parallel level p increases from 1 to 32, while the consumed area of the straightforward version on ASIC can increase from 5975 *GE* to 13381 *GE* as the parallel level p increases from 1 to 32. Because the increase rate of

Table 2: Evaluation results on ASIC.

p	Period (ns)	Area (GE)	Throughput (Gbps)	Efficiency (Mbps/GE)	Technique
x1	0.44	5975	1.14	0.19	Straightforward
x2	0.46	5705	2.17	0.38	
x4	0.46	6678	4.35	0.65	
x8	0.48	7676	8.33	1.09	
x16	0.57	8692	14.55	1.61	
x32	0.60	13381	26.67	1.99	
x1	0.41	6028	1.22	0.20	Galois transformation
x2	0.41	6399	2.44	0.38	
x4	0.44	6742	4.55	0.66	
x8	0.45	7744	8.89	1.15	
x16	0.47	10194	17.39	1.67	
x1	0.42	6192	1.19	0.19	Pipeline-like pre-computation
x2	0.41	6419	2.44	0.38	
x4	0.42	6877	4.76	0.69	
x8	0.43	8116	9.30	1.15	
x16	0.44	10559	18.18	1.72	
x32	0.49	15249	32.65	2.14	
x1	0.47	10415	1.06	0.10	Masking straightforward
x2	0.48	11192	0.69	0.06	
x4	0.49	13701	1.36	0.10	
x8	0.57	17471	2.34	0.13	
x16	0.61	25788	4.37	0.17	
x32	0.67	44246	7.96	0.18	
x1	0.43	11615	1.16	0.10	Masking pipeline-like pre-computation
x2	0.44	13305	2.27	0.17	
x4	0.45	16866	4.44	0.26	
x8	0.48	22722	8.33	0.37	

Table 3: Evaluation results on FPGA.

p	Period (ns)	Area			Throughput (Gbps)	Efficiency (Mbps/Slice)	Technique
		(Slice)	(LUT)	(FF)			
x1	2.76	158	403	629	0.18	1.15	Straightforward
x2	2.81	168	463	625	0.36	2.12	
x4	2.85	191	547	623	0.70	3.67	
x8	2.90	201	634	625	1.38	6.86	
x16	3.08	355	1073	615	2.60	7.32	
x32	4.04	502	1730	614	3.96	7.89	
x1	2.68	185	611	632	0.19	1.01	Galois transformation
x2	2.72	214	669	627	0.37	1.72	
x4	2.78	224	779	627	0.72	3.21	
x8	2.80	274	853	632	1.43	5.21	
x16	2.91	341	1148	703	2.75	8.06	
x1	2.67	212	623	632	0.19	0.88	Pipeline-like pre-computation
x2	2.71	214	694	632	0.37	1.72	
x4	2.76	236	794	635	0.72	3.07	
x8	2.80	263	907	640	1.43	5.43	
x16	2.89	371	1216	719	2.77	7.46	
x32	3.19	537	1849	760	5.02	9.34	
x1	2.81	210	683	1098	0.06	0.28	Masking straightforward
x2	2.90	350	1286	1177	0.11	0.33	
x4	3.01	464	1565	1337	0.22	0.48	
x8	3.22	593	1888	1648	0.41	0.70	
x16	3.66	840	2681	2278	0.73	0.87	
x32	4.71	1176	4139	3513	1.13	0.96	
x1	2.93	226	774	1104	0.17	0.76	Masking pipeline-like pre-computation
x2	2.94	414	1428	1195	0.34	0.82	
x4	3.01	506	1875	1370	0.66	1.31	
x8	3.24	738	2643	1718	1.28	1.72	

the throughput (25.6 times) can be larger than the increase rate of the consumed area (1.2 times), the throughput-area ratio of the straightforward version on ASIC can increase with the parallel level p. Similarly, the throughput of the straightforward version on FPGA can increase from 0.18 Gbps to 3.96 Gbps as the parallel level p increases from 1 to 32, while the consumed area of the straightforward version on FPGA can increase from 158 Slices to 502 Slices as the parallel level p increases from 1 to 32. Because the increase rate of the throughput (21 times) can be larger than the increase rate of the consumed area (2.2 times), the throughput-area ratio of the straightforward version on FPGA can also increase with the parallel level p. The Galois transformation version and the pipeline-like pre-computation version can show similar trends on ASIC and FPGA.

(iii) Third, compared with the hardware implementation of Grain-128AEADv2, the masked hardware implementation of Grain-128AEADv2 can decrease the throughput and increase the consumed area. Accordingly, the throughput-area ratio of the masked hardware implementation of Grain-128AEADv2 can be lower than that of the hardware implementation of Grain-128AEADv2. Comparatively, the decrease rate of the throughput-area ratio of the pipeline-like pre-computation version of the masked hardware implementation of Grain-128AEADv2 to the throughput-area ratio of the pipeline-like pre-computation version of the hardware implementation of Grain-128AEADv2 can be smaller than the decrease rate of the throughput-area ratio of the straightforward version of the masked hardware implementation of Grain-128AEADv2 to the throughput-area ratio of the straightforward version of the hardware implementation of Grain-128AEADv2. In detail, for the straightforward version, the highest throughput on ASIC can decrease from 26.67 *Gbps* to 7.96 *Gbps* and the largest consumed area can increase from 13381 *GE* to 44246 *GE*, while for the pipeline-like pre-computation version, the highest throughput on ASIC can decrease from 32.65 *Gbps* to 8.33 *Gbps* and the largest consumed area can increase from 15249 *GE* to 22722 *GE*. Then, for the straightforward version, the highest throughput-area ratio on ASIC can decrease from 1.99 Mbps/*GE* to 0.18 Mbps/*GE*, while for the pipeline-like pre-computation version, the highest throughput-area ratio on ASIC can decrease from 2.14 Mbps/*GE* to 0.37 Mbps/*GE*. Therefore, the highest throughput-area ratio of the straightforward version of the masked hardware implementation of Grain-128AEADv2 on ASIC can decrease about 90% compared with the highest throughput-area ratio of the straightforward version of the hardware implementation of Grain-128AEADv2 on ASIC, while the highest throughput-area ratio of the pipeline-like pre-computation version of the masked hardware implementation of Grain-128AEADv2 on ASIC can decrease about 80% compared with the highest throughput-area ratio of the pipeline-like pre-computation version of the hardware implementation of Grain-128AEADv2 on ASIC. Such trend can also be shown on FPGA. In summary, since the increase rate of the consumed area of the pipeline-like pre-computation version can be smaller than the increase rate of the consumed area of the straightforward version while the decrease rate of the throughput of two versions can be about the same, we obtain the result that the decrease rate of the throughput-area ratio of the pipeline-like pre-computation version can be smaller than that of the throughput-area ratio of the straightforward version.

6. Security Evaluation

In this section, the resistance of the masked hardware implementation of Grain-128AEADv2 against side-channel attack is evaluated with T-Test proposed by Gilbert Goodwill et al. [22] in the simulated scenario. More specifically, the *non-specific* T-Test leakage detection methodology is adopted. In the non-specific T-Test detection methodology, two sets of power traces should be used. Power traces in one set Q_1 correspond to the encryption of randomly chosen IVs with a fixed secret key, while power traces in another set Q_2 correspond to the encryption of a fixed IV with the same fixed secret key. If the number of samples contained in one power trace is denoted as N_s, the value v of T-Test at sample $w(1 \le w \le N_s)$ can be computed as

$$v_w = \frac{X_{1,w} - X_{2,w}}{\sqrt{S_{1,w}^2/N_1 + S_{2,w}^2/N_2}}, \tag{29}$$

where $X_{1,w}$ denotes the mean of the power traces contained in Q_1 at sample w, $X_{2,w}$ denotes the mean of the power traces contained in Q_2 at sample w, $S_{1,w}^2$ denotes the variance of the power traces contained in Q_1 at sample w, $S_{2,w}^2$ denotes the variance of the power traces contained in Q_2 at sample w, N_1 denotes the number of power traces contained in Q_1, and N_2 denotes the number of power traces contained in Q_2. The null hypothesis is that $X_{1,w}$ and $X_{2,w}$ can be equal, which is accepted if v_w is between the threshold of ±4.5. If v_w exceeds the threshold of ±4.5, the null hypothesis is rejected with a confidence greater than 99.999%. In the simulated scenario, the power consumption in a power trace at sample w is assumed to be composed of the signal part and the noise part. The signal part is simulated under the *Hamming Distance Model*, while the noise part is assumed to follow the *Gaussian Distribution* with mean 0 and a given variance σ^2.

We observe that the variance of the signal under the Hamming Distance Model can be between 24 and 66. Then, the signal-to-noise ratio of the hardware implementation is set to 0.02. According to the evaluation results, the leakage of

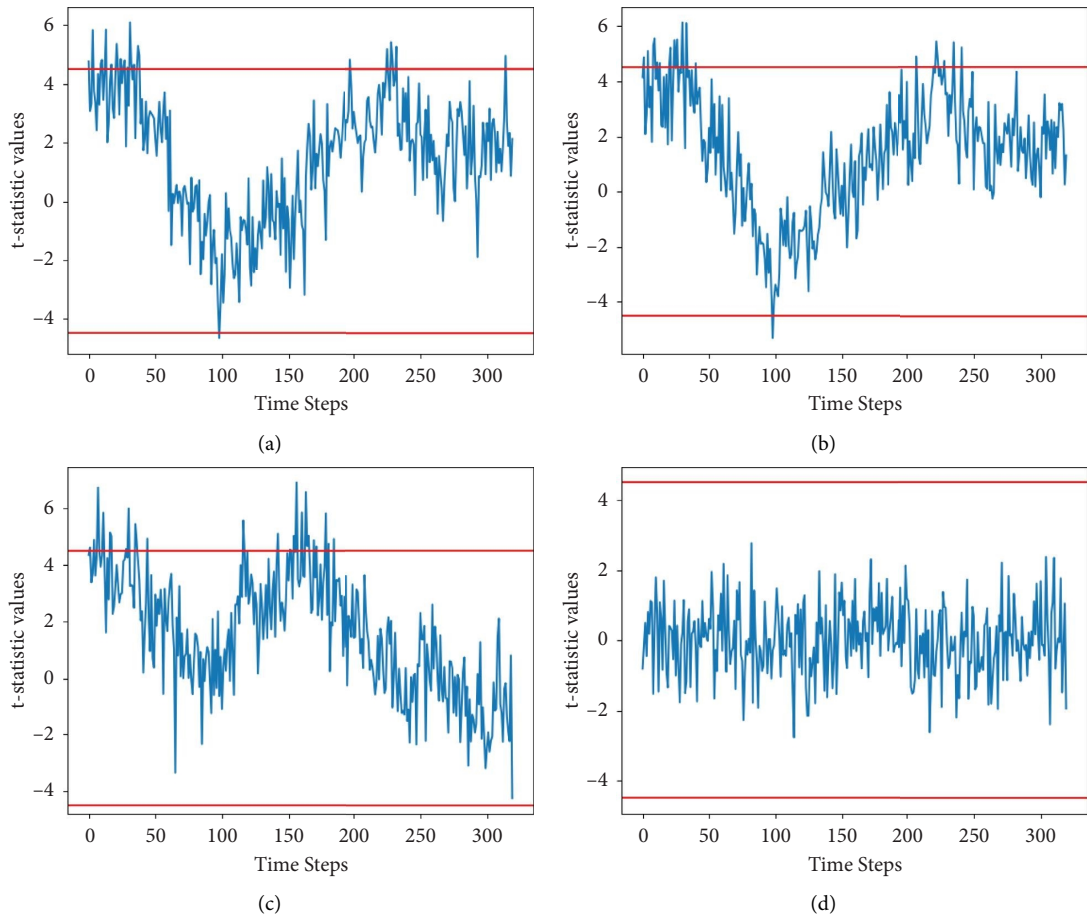

Figure 8: The security evaluation results of the (masked) hardware implementation of Grain-128AEADv2. (a) Straightforward. (b) Pipeline-like pre-computation. (c) Galois transformation. (d) Masking.

an unprotected hardware implementation can be tested with only 10,000 traces. The number of traces in the simulated scenario is less than that needed in the real scenario. The reason can be as follows. In the simulated scenario, the signal leakage can perfectly follow the Hamming Distance Model, while in the real scenario, it is impossible to perfectly characterize the signal leakage. Thus, much more traces can be needed in the real scenario. Note that 100,000 traces are used to test the leakage of the hardware implementation of Grain-128AEADv2. The security evaluation results of the (masked) hardware implementation of Grain-128AEADv2 are shown in Figure 8.

In Figure 8, the red lines represent the threshold of ±4.5. If the T-Test value exceeds ±4.5, the leakage of the hardware implementation of Grain-128AEADv2 can be tested; otherwise, no leakage can be tested. According to Figure 8, the following three observations can be obtained.

(i) First, the leakage of the unprotected hardware implementation of Grain-128AEADv2 with either the straightforward technique, the Galois transformation technique, or the pipeline-like pre-computation technique can be tested with 100,000 traces, which means that three types of unprotected hardware implementations of Grain-128AEADv2 can be insecure against side-channel attacks.

(ii) Second, the T-Test values of the masked hardware implementation of Grain-128AEADv2 with the pipeline-like pre-computation technique can be within the threshold of ±4.5, which means that such hardware implementation of Grain-128AEADv2 can be secure in face of side-channel attacks. Therefore, the security evaluation results show the effectiveness of masking against side-channel attacks.

(iii) Third, one can see that the shape of two curves of T-Test computed for the straightforward version and the pipeline-like pre-computation version can be about the same, while that computed for the Galois transformation version can be different. The reason is that the pipeline-like pre-computation version does not change the state of the FSRs and only adds two registers to store the intermediate values. The effect of the two extra registers can be ignored compared to the states of FSRs, which induces the result that the shape of the T-Test curve of the pipeline-like pre-computation version can be about the same with the shape of the T-Test curve of

the straightforward version. However, the Galois transformation version can change the state of the NFSR, which induces the result that the shape of the T-Test curve of the Galois transformation version can be different from the shape of the T-Test curve of the straightforward version.

7. Conclusion

In this paper, efficient (masked) hardware implementation of Grain-128AEADv2 is considered and we propose the pipeline-like pre-computation technique. For the unmasked version, the hardware implementation of Grain-128AEADv2 with the straightforward technique, the Galois transformation technique, and the pipeline-like pre-computation technique is considered. For the masked version, the hardware implementation of Grain-128AEADv2 with the straightforward technique and the pipeline-like pre-computation technique is considered. The performance of the (masked) hardware implementation of Grain-128AEADv2 is evaluated on both ASIC and FPGA. According to the evaluation results, the pipeline-like pre-computation technique can optimize the throughput-area ratio for the masked version and the unmasked version compared with the other two techniques. In detail, the highest throughput-area ratio of the hardware implementation of Grain-128AEADv2 can be obtained in the x32 parallel version with the pipeline-like pre-computation technique, which is 2.14 Mbps/GE on ASIC and 9.34 Mbps/Slice on FPGA; the highest throughput-area ratio of the masked hardware implementation of Grain-128AEADv2 can be obtained in the x8 parallel version with the pipeline-like pre-computation technique, which is 0.37 Mbps/GE on ASIC and 1.72 Mbps/Slice on FPGA. Besides, the security of the masked hardware implementation of Grain-128AEADv2 with the pipeline-like pre-computation technique against side-channel attacks is evaluated with T-Test in simulated scenarios. Overall, this contribution may help researchers and practitioners to accurately compare the efficiency and the security of the hardware implementation of Grain-128AEADv2 with those of other lightweight cryptographic algorithms.

Data Availability

All data generated or analyzed during this study are included in this article.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

Acknowledgments

This study was supported by the National Key Research and Development Program of China (no. 2020YFB1805402), the Open Fund of Advanced Cryptography and System Security Key Laboratory of Sichuan Province (grant no. SKLACSS-202116), and the National Natural Science Foundation of China (grant nos. 61872359, 61936008, and 62272451).

References

[1] Ecrypt, "eSTREAM: Ecrypt Stream Cipher Project," 2008, http://www.ecrypt.eu.org/stream.

[2] M. Hell, T. Johansson, A. Maximov, and W. Meier, "The grain family of stream ciphers," in *New Stream Cipher Designs*, pp. 179–190, Springer, 2008.

[3] M. Hell, T. Johansson, A. Maximov, and W. Meier, "A stream cipher proposal: grain-128," in *Proceedings of the 2006 IEEE International Symposium on Information Theory*, pp. 1614–1618, IEEE, Seattle, WA, USA, July 2006.

[4] M. Ågren, M. Hell, T. Johansson, and W. Meier, "Grain-128 a: a new version of grain-128 with optional authentication," *International Journal of Wireless and Mobile Computing*, vol. 5, no. 1, pp. 48–59, 2011.

[5] M. Hell, T. Johansson, A. Maximov, W. Meier, and H. Yoshida, "Grain-128aead - a lightweight aead stream cipher," *NIST Lightweight Cryptography*, NIST, Gaithersburg, MD, USA, 2019, https://csrc.nist.gov/Projects/lightweight-cryptography/round-1-candidates.

[6] M. Hell, T. Johansson, A. Maximov, W. Meier, and H. Yoshida, "Grain-128aeadv2: Strengthening the initialization against key reconstruction," *Cryptology ePrint Archive*, NIST, Gaithersburg, MD, USA, 2021, https://eprint.iacr.org/2021/751.

[7] S. S. Mansouri and E. Dubrova, "An improved hardware implementation of the grain-128a stream cipher," in *Information Security and Cryptology – ICISC 2012*, T. Kwon, M. K. Lee, and D. Kwon, Eds., pp. 278–292, Springer, Berlin, Heidelberg, 2013.

[8] J. Sönnerup, M. Hell, M. Sönnerup, and R. Khattar, "Efficient hardware implementations of grain-128aead," in *Progress in Cryptology – INDOCRYPT 2019*, F. Hao, S. Ruj, and S. Sen Gupta, Eds., pp. 495–513, Springer International Publishing, New York, NY, USA, 2019.

[9] E. Dubrova, "Finding matching initial states for equivalent nlfsrs in the fibonacci and the galois configurations," *IEEE Transactions on Information Theory*, vol. 56, no. 6, pp. 2961–2966, 2010.

[10] J. S. Coron and L. Goubin, "On boolean and arithmetic masking against differential power analysis," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 231–237, Springer, Berlin, Heidelberg, 2000.

[11] Y. Ishai, A. Sahai, and D. Wagner, "Private circuits: securing hardware against probing attacks," in *Annual International Cryptology Conference*, pp. 463–481, Springer, Berlin, Heidelberg, 2003.

[12] N. Veyrat-Charvillon, M. Medwed, S. Kerckhof, and F. X. Standaert, "Shuffling against side-channel attacks: a comprehensive study with cautionary note," in *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 740–757, Springer, New York, NY, USA, 2012.

[13] J. S. Coron and I. Kizhvatov, "An efficient method for random delay generation in embedded software," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 156–170, Springer, Berlin, Heidelberg, 2009.

[14] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *Annual International Cryptology Conference*, pp. 398–412, Springer, Berlin, Heidelberg, 1999.

[15] S. Mangard, T. Popp, and B. M. Gammel, "Side-channel leakage of masked cmos gates," in *Topics in Cryptology – CT-RSA 2005*, A. Menezes, Ed., pp. 351–365, Springer, Berlin, Heidelberg, 2005.

[16] S. Mangard, N. Pramstaller, and E. Oswald, "Successfully attacking masked aes hardware implementations," in *Cryptographic Hardware and Embedded Systems – CHES 2005*, J. R. Rao and B. Sunar, Eds., pp. 157–171, Springer, Berlin, Heidelberg, 2005.

[17] S. Mangard and K. Schramm, "Pinpointing the side-channel leakage of masked aes hardware implementations," in *Cryptographic Hardware and Embedded Systems - CHES 2006*, L. Goubin and M. Matsui, Eds., pp. 76–90, Springer, Berlin, Heidelberg, 2006.

[18] O. Reparaz, B. Bilgin, S. Nikova, B. Gierlichs, and I. Verbauwhede, "Consolidating masking schemes," in *Annual Cryptology Conference*, pp. 764–783, Springer, Berlin, Heidelberg, 2015.

[19] S. Faust, V. Grosso, S. Merino Del Pozo, C. Paglialonga, and F. X. Standaert, "Composable masking schemes in the presence of physical defaults & the robust probing model," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 89–120, 2018.

[20] H. Groß, S. Mangard, and T. Korak, "Domain-oriented masking: compact masked hardware implementations with arbitrary protection order," in *TIS@ CCS*p. 3, IAIK, Graz, Austria, 2016.

[21] H. Gross and S. Mangard, "A unified masking approach," *Journal of cryptographic engineering*, vol. 8, no. 2, pp. 109–124, 2018.

[22] B. J. Gilbert Goodwill, J. Jaffe, and P. Rohatgi, *A testing methodology for side-channel resistance validation*, vol. 7, NIST non-invasive attack testing workshop, Gaithersburg, MD, USA, 2011.

[23] K. Jens-Peter, D. William, T. Michael, H. Ekawat, and G. Kris, "Hardware api for lightweight cryptography v1.1 (with support for sca-protected implementations)," 2022, https://cryptography.gmu.edu/athena/LWC/LWC_HW_API.pdf.

[24] M. Kamyar, T. Michael, F. Farnoud et al., "Implementer's guide to hardware implementations compliant with the hardware api for lightweight cryptography v1.2.0," 2022, https://cryptography.gmu.edu/athena/LWC.

[25] Xilinx, "7 series fpgas data sheet: overview," 2020, https://docs.xilinx.com/v/u/en-US/ds180_7Series_Overview.

[26] Xilinx, "Vivado design suite - hlx editions," 2023, https://www.xilinx.com/support/documentation-navigation/development-tools/hardware-development/vivado-design-suite.html.