

Article

Design and Analysis of Area and Energy Efficient Reconfigurable Cryptographic Accelerator for Securing IoT Devices

Xvpeng Zhang ^{1,†}, Bingqiang Liu ^{1,†} , Yaqi Zhao ¹, Xiaoyu Hu ¹, Zixuan Shen ¹, Zhaoxia Zheng ^{1,2}, Zhenglin Liu ^{1,2}, Kwen-Siong Chong ^{3,4}, Guoyi Yu ^{1,2,*}, Chao Wang ^{1,2,*}  and Xuecheng Zou ^{1,2}

¹ School of Optical and Electronic Information, Huazhong University of Science and Technology, Wuhan 430074, China

² Wuhan National Laboratory of Optoelectronics, Huazhong University of Science and Technology, Wuhan 430074, China

³ Temasek Laboratories, Nanyang Technological University, Singapore 639798, Singapore

⁴ Zero-Error Systems Pte Ltd., TCH Techcentre, 71 Toh Guan Road East, #05-07, Singapore 608598, Singapore

* Correspondence: yuguoyi@hust.edu.cn (G.Y.); chao_wang_me@hust.edu.cn (C.W.)

† These authors contributed equally to this work.

Abstract: Achieving low-cost and high-performance network security communication is necessary for Internet of Things (IoT) devices, including intelligent sensors and mobile robots. Designing hardware accelerators to accelerate multiple computationally intensive cryptographic primitives in various network security protocols is challenging. Different from existing unified reconfigurable cryptographic accelerators with relatively low efficiency and high latency, this paper presents design and analysis of a reconfigurable cryptographic accelerator consisting of a reconfigurable cipher unit and a reconfigurable hash unit to support widely used cryptographic algorithms for IoT Devices, which require block ciphers and hash functions simultaneously. Based on a detailed and comprehensive algorithmic analysis of both the block ciphers and hash functions in terms of basic algorithm structures and common cryptographic operators, the proposed reconfigurable cryptographic accelerator is designed by reusing key register files and operators to build unified data paths. Both the reconfigurable cipher unit and the reconfigurable hash unit contain a unified data path to implement Data Encryption Standard (DES)/Advanced Encryption Standard (AES)/ShangMi 4 (SM4) and Secure Hash Algorithm-1 (SHA-1)/SHA-256/SM3 algorithms, respectively. A reconfigurable S-Box for AES and SM4 is designed based on the composite field Galois field (GF) $GF(((2^2)^2)^2)$, which significantly reduces hardware overhead and power consumption compared with the conventional implementation by look-up tables. The experimental results based on 65-nm application-specific integrated circuit (ASIC) implementation show that the achieved energy efficiency and area efficiency of the proposed design is 441 Gbps/W and 37.55 Gbps/mm², respectively, which is suitable for IoT devices with limited battery and form factor. The result of delay analysis also shows that the number of delay cycles of our design can be reduced by 83% compared with the state-of-the-art design, which shows that the proposed design is more suitable for applications including 5G/Wi-Fi/ZigBee/Ethernet network standards to accelerate block ciphers and hash functions simultaneously.

Keywords: reconfigurable cryptographic accelerator; hardware security; intelligent sensors and mobile robots; intelligent Internet of Things



Citation: Zhang, X.; Liu, B.; Zhao, Y.; Hu, X.; Shen, Z.; Zheng, Z.; Liu, Z.; Chong, K.-S.; Yu, G.; Wang, C.; et al. Design and Analysis of Area and Energy Efficient Reconfigurable Cryptographic Accelerator for Securing IoT Devices. *Sensors* **2022**, *22*, 9160. <https://doi.org/10.3390/s22239160>

Academic Editor: Gianni D'Angelo

Received: 1 November 2022

Accepted: 22 November 2022

Published: 25 November 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Internet of Things (IoT) devices, including intelligent sensors and mobile robots, often collect information from the environment, and transmit a large amount of data to the cloud and other IoT devices. This large amount of data needs to be managed, processed, transferred and stored securely [1,2]. To enable IoT end-to-end security, cryptographic

primitives such as block ciphers and/or hash functions in the IoT devices need to support multiple network communication standards (such as 5G, Wi-Fi, ZigBee, and Ethernet [3]), to ensure communication privacy and integrity [4], and to abide by various security protocols regulated by various security standards and countries. The block ciphers include Data Encryption Standard (DES), Advanced Encryption Standard (AES) and ShangMi 4(SM4), whereas the hash functions include Secure Hash Algorithm-1 (SHA-1), SHA-256 and SM3 [5,6]. Although these various cryptographic primitives can easily be implemented at the software level [7] (e.g., by using a microcontroller), unfortunately the software implementation often results in high latency and large energy consumption overheads, hence limiting deployment on IoT devices which are highly resource constrained.

To mitigate the high latency and large energy consumption overheads, cryptographic accelerators for various cryptographic primitives are often adopted. The common practice includes the realization of a dedicated cryptographic accelerator for one specific cryptographic primitive. However, such a practice cannot support multiple cryptographic primitives as required by the emerging security protocols for IoT devices [8,9]. The direct implementation of each cryptographic primitive, with its dedicated cryptographic accelerator, causes large hardware overhead and energy consumption, which is similarly not an efficient solution for resource-constrained IoT devices. Alternatively, the idea of configuring cryptographic accelerators to support multiple cryptographic primitives has been reported. Liu et al. [10] have reported a dynamically reconfigurable processing array for supporting DES/AES/SM4 and SHA-256/SM3. Du et al. [11] have reported a coarse-grained reconfigurable cryptographic logic array and an intelligent mapping algorithm for different cryptographic algorithms, such as DES/AES/SM4/SHA-256, etc. Deng et al. [12] have reported a coarse-grained reconfigurable cryptographic processor, which can implement all the commonly used cryptographic algorithms, including DES, AES, SM4, SHA-256, SM3, etc.

These reported configurable cryptographic accelerators, however, remain inefficient for three reasons. First, they adopt a unified hardware unit approach by integrating both block ciphers and hash functions into a common unit, and are hence unable to process both block cipher and hash functions simultaneously. Considering some actual application scenarios, the protocols, including the Wi-Fi Protected Access (WPA) widely used in Wi-Fi, and the Datagram Transport Layer Security (DTLS)/Internet Protocol Security (IPsec) protocol widely used in ethernet standards [7,9,10], require both encryption and hash functions to be performed simultaneously. The series processing of the unified hardware unit increases latency. Second, the unified hardware unit is not circuit optimized because it has to accommodate distinct differences in terms of data paths and logic operations between the block ciphers and hash functions. Third, they adopt look-up tables (LUTs) to implement the substitution box (S-Box), which causes large hardware overhead and is therefore not an efficient solution.

In this paper, a reconfigurable cryptographic accelerator is proposed that has three-tier configurability (i.e., functional configurability, data-path configurability, and S-Box configurability) to support widely used security functions for resource-limited and energy-constrained IoT devices including small mobile intelligent robots, wearable medical devices and miniature unmanned aerial vehicles. The major contributions of the paper are as follows.

1. At tier1 of the functional configurability, the proposed reconfigurable cryptographic accelerator can perform either block ciphers (DES/AES/SM4) alone, or hash functions (SHA-1/SHA-256/SM3) alone or both block cipher and hash function simultaneously. This is achieved by having two separate unified reconfigurable engines (i.e., one for the block ciphers, and another for the hash functions), that account for latency optimization for actual application scenarios including authentication and encryption/decryption processes.

2. At tier2 of the datapath configurability, the datapath for the block ciphers and for the hash functions are optimized. This is achieved by first having a detailed and comprehensive algorithmic analysis of both the block ciphers and hash functions in terms of basic algorithm structures and common cryptographic operators to guide efficient reconfigurable architecture design, which provides a practical design reference for the reconfigurable hardware design of cryptographic algorithms. Through this analysis, in the proposed design implementation, the datapath for the block ciphers is optimized to have a 33.65% smaller area, and the datapath for the hash functions a 56.18% smaller area.
3. At tier3 of the S-Box configurability, a reconfigurable S-Box module for AES and SM4 is designed based on the composite Galois field (GF) $GF(((2^2)^2)^2)$ instead of $GF(2^8)$, which significantly reduces hardware overhead by more than 30% as compared to the conventional implementations of LUTs.

The remainder of this paper is organized as follows: Section 2 provides an algorithm analysis of both block ciphers and hash functions. Section 3 describes the architecture and hardware implementation of the proposed reconfigurable cryptographic accelerator. Section 4 presents experimental results and discussions. Section 5 presents the conclusion.

2. Algorithm Analysis of Block Ciphers and Hash Functions

This section first provides a brief review of the application scenarios of block ciphers and hash functions and then presents an introduction to the structures and principles of the cryptographic algorithms.

2.1. Review of Cryptographic Algorithms in IoT Security Applications

Edge IoT devices face the security threat of data disclosure during communication. Various IoT network standards that apply different cryptographic algorithms are an important means to ensure communication security. For communication security, ensuring the privacy and integrity of transmitted data is a basic requirement. Block ciphers and hash functions are two categories of major cryptographic algorithms that have been widely used in various IoT network standards to ensure data privacy and integrity, respectively. Block cipher is a symmetric cryptographic algorithm that performs encryption or decryption on a fixed-size data block using a shared secret key. Plaintext is the source data during the encryption, and the resulting encrypted text is called ciphertext. The same key is used for both the encryption of plaintext and the decryption of ciphertext. Hash function is a type of security mechanism that produces a hash value, message digest or checksum value for a specific data object. If the data are intentionally or unintentionally modified, the hash value is changed. Thus, the integrity of data objects can be evaluated by comparing and verifying previous and current hash values of the transmitted data.

Table 1 shows major algorithms of block ciphers and hash functions that have been widely used in different security protocols and communication standards. It can be found that DES/AES/SM4 and SHA-1/SHA-256/SM3 are the two major algorithm groups for block ciphers and hash functions, respectively, which are widely deployed in major countries/regions for various network application scenarios. Notably, some of the key cryptographic algorithms need to be used at the same time. For example, DES/AES/SM4 and SHA-1/SHA-256/SM3 are simultaneously used to perform encryption and authentication, respectively, as part of the Internet Protocol Security (IPsec), which is a set of protocols widely used for the secure establishment of virtual private networks (VPNs). Similarly, AES/DES/SM4 and SHA-1/SHA-256/SM3 are also simultaneously used in the Transport Layer Security (TLS), which is a security protocol used to secure data transmitted over a network by preventing data from being eavesdropped on or tampered with. These security protocols, using different cryptographic algorithms, are able to support various communication standards, including 5G, Wi-Fi, ZigBee, and Ethernet, which are the key to achieving seamless network communication among the IoT devices in the different networks deployed in different countries and regions. Therefore, DES/AES/SM4 and

SHA-1/SHA-256/SM3 algorithms were selected in this study for reconfigurable hardware accelerator design, which is able to effectively support IoT security for major countries and regionstitute of Standards and Technology.

Table 1. Major algorithms in block ciphers and hash functions.

	Algorithms	Security Protocols	Applicable Countries/Regions	Organization	Typical Application Example
Block Cipher	DES	IPsec/SSL/TLS	USA/CHN/EUR	NIST	Ethernet
	AES	Most communication standards/protocols	USA/CHN/EUR	NIST	WLAN/ZigBee/Bluetooth/5G/LTE/Ethernet
	SM4	IPsec/TLS/DTLS/SSL/WAPI	CHN	SCA	WLAN/Ethernet
Hash function	SHA-1	IPsec/SSL/TLS	USA/CHN/EUR	NIST	RFID/Ethernet
	SHA-256	Most communication standards/protocols	USA/CHN/EUR	NIST	WLAN/Bluetooth/5G/LTE/Ethernet
	SM3	IPsec/SSL/TLS	CHN	SCA	WLAN/Ethernet

NIST: National Institute of Standards and Technology. SCA: State Cryptography Administration.

2.2. Introduction of Major Block Ciphers and Hash Functions

2.2.1. Algorithm Analysis of Major Block Ciphers

Block ciphers usually utilize multiple iterations of round operations on data blocks to implement confusion and diffusion functions that ensure strong data security. There are two major constructions of round operations in block ciphers: Feistel networks and Substitution–Permutation (SP) network [13,14].

Figure 1a presents a Feistel network containing n same-round operations. Each round operation contains a key-schedule operation generating round key K_i ($i \in \{1, 2, \dots, n\}$), a round function F and a consequent XOR operation. The inputs of the round function F are data block $X_{i \text{ right}}$ from the former round operation (excluding the initial plaintext input $X_{0 \text{ right}}$ of the first-round operation) and the round key K_i . The round function F contains basic operations including addition, S-Box, shift, and permutation to realize the confusion and diffusion of plaintext data, which have been adopted in the DES/SM4 algorithms. The purpose of confusion is to make the statistical relationship between plaintext and ciphertext as complex as possible, while diffusion is used to quickly spread the statistical characteristics of the plaintext into the ciphertext. The output of the round function F is XORed with the remaining data block $X_{i \text{ left}}$ from the former round operation (excluding the initial plaintext input $X_{0 \text{ left}}$ of the first-round operation). After n same-round operations, the final output is the ciphertext. It is worth noting that the Feistel network-based encryption and decryption have the same operation structure, and the only difference is that the round keys are applied in the reverse order in the decryption process as shown in Figure 1a.

Figure 1b shows the construction of an SP network also containing n same round operations. Each round operation applies several substitution boxes (S-Boxes), a permutation box (P-Box) and a consequent XOR operation on the plaintext block to produce the ciphertext block, which has been used in the AES algorithms. The input of the S-Boxes in each round operation is from the former round operation (excluding the first-round operation taking the XOR of plaintext and the initial round key K_0 as input). The S-Boxes and P-box can efficiently realize confusion and diffusion of plaintext data, respectively. The S-Boxes realize the nonlinear permutation of block data, while a P-box realizes the permutation of all the data bits, which takes the outputs of all former S-Boxes in the same-round operation and permutes the data bits. The output of the P-box is XORed with the round key K_i generated by the key-schedule operation to complete one round operation. Similarly, after n same-round operations, the final output is the ciphertext. It is worth noting that the decryption of the SP-based algorithm is the inverse operation of the encryption process as shown in Figure 1b.

Figure 2a shows the data flow of the DES algorithm with a 64-bit input block, a 64-bit key size and a 64-bit output block. For the encryption process, there is an initial permutation, 16 round operations and a final permutation. After the initial permutation, the 64-bit data block is divided into two 32-bit halves ($X_{0\text{ left}}$ and $X_{0\text{ right}}$) and subsequently processed by the Feistel round operators in a crisscrossing manner. There are two major parts in each round operation, i.e., the key-schedule part and the Feistel function part, as shown in Figure 2a. For the key-schedule part, the 56 bits of the key are initially selected from the initial 64 bits by permuted choice 1, divided into two 28-bit halves, and then rotated left by one or two bits (specified for each round) in successive rounds. Next, 48-bit round key are selected from the 56-bit input by permuted choice 2 with 24 bits from the left, and 24 bits from the right. For the Feistel function part, there are four steps, including the extended permutation, key mixing, substitution and P-box permutation. In the extended permutation, the 32-bit right half-block ($X_{n\text{ right}}$) is expanded to 48 bits using the expansion permutation. In the key mixing, the 48 bits from extended permutation are XORed with the key in this round. In the substitution, the 48-bit block is divided into eight 6-bit blocks and eight S-Boxes are used to replace the 6-bit input with 4-bit output according to a non-linear transformation. In the P-box permutation, the eight 4-bit outputs from the eight S-Boxes are rearranged according to a P-box permutation to achieve diffusion as mentioned before. The 32-bit output of the Feistel function part is XORed with the 32-bit left half-block ($X_{n\text{ left}}$) to get the final result of one round operation. For the decryption process, the only difference from the encryption process is that the round keys are applied in reverse order in the decryption process.

Figure 2b shows the data flow of the AES-128 algorithm with a 128-bit input block, a 128-bit key and a 128-bit output block as an example to illustrate the AES algorithms with 128/192/256-bit keys. As compared to the AES-128 algorithm with 10-round operations, the AES-192 and AES-256 algorithms also process on a block of 128-bit data but with 192-bit and 256-bit keys, respectively, which corresponds to 12 and 14 rounds of operations, respectively. Different from the DES algorithm, the AES uses the SP structure instead of the Feistel structure. For the encryption process, there are four major steps, i.e., SubBytes, ShiftRows, MixColumns and AddRoundKey, in one round operation. In the SubBytes step, each byte X_i ($i = 0-15$) is permuted using an 8-bit S-Box and there are a total 16 S-Boxes used in this step. In the ShiftRows step, data bytes in each row are shifted to the left, cyclically excluding the first row. In the MixColumns step, each column is multiplied with a fixed matrix. In the AddRoundKey step, each byte is XORed with a byte of the round key. For the round key schedule process, the right column (K_0, K_1, K_2, K_3) is first cyclically shifted to the left with one byte. Then four S-Boxes are used to perform byte substitution. Next, the results from the four S-Boxes and R_{con} are XORed with the left column ($K_{12}, K_{13}, K_{14}, K_{15}$) to get a new column ($K'_{12}, K'_{13}, K'_{14}, K'_{15}$). Finally, the remaining three columns of round keys can be obtained as shown in Figure 2b. It is to be noted that the last round operation does not include the MixColumns step. For the decryption process, InvShiftRows (the inverse of ShiftRows), InvSubBytes (the inverse of SubBytes), AddRoundKey (XOR operation using the inverse order of round key) and InvMixColumns (the inverse of MixColumns) are used in each round operation.

Figure 2c shows the data flow of the SM4 algorithm with a 128-bit input block, a 128-bit key and a 128-bit output block, which is the same as the AES-128. There are 32 round operations to calculate the final ciphertext. Similar to the DES, the SM4 also uses the Feistel structure in each round operation and there are also a key-schedule and a Feistel function part. For the key-schedule part, the 128-bit original key is XORed with fixed parameter FK, and then the result is divided into four 32-bit data, i.e., K_0, K_1, K_2 and K_3 . The K_0, K_1 and K_2 are XORed with a 32-bit fixed parameter R_{con} , and then the result is divided into four 8-bit data, which are input to four S-Boxes. The four results from four S-Boxes are combined into one 32-bit datum, which is shifted left cyclically by 13 and 23 bits. The original and shifted data are XORed with K_3 to get K_0 in the next round as shown in Figure 2c. Similar to the key-schedule part, for the Feistel function part, the 128-bit block is divided into

four 32-bit data, i.e., X_0 , X_1 , X_2 and X_3 . The X_0 , X_1 and X_2 are first XORed with a 32-bit round key K_0 to get a 32-bit result, which is divided into four 8-bit data and input into four S-Boxes to perform the substitution. The four results from S-Boxes are combined into a 32-bit output data, which will be shifted left cyclically by 2, 10, 18 and 24 bits respectively to obtain four 32-bit data. The four data are XORed with the original data to get one result to complete the Feistel operation. Finally, X_3 is XORed with the 32-bit Feistel result to get X_0 in the next round as shown in Figure 2c. For the decryption process, similar to DES, the only difference from the encryption process is that the round keys are applied in the reverse order in the decryption process.

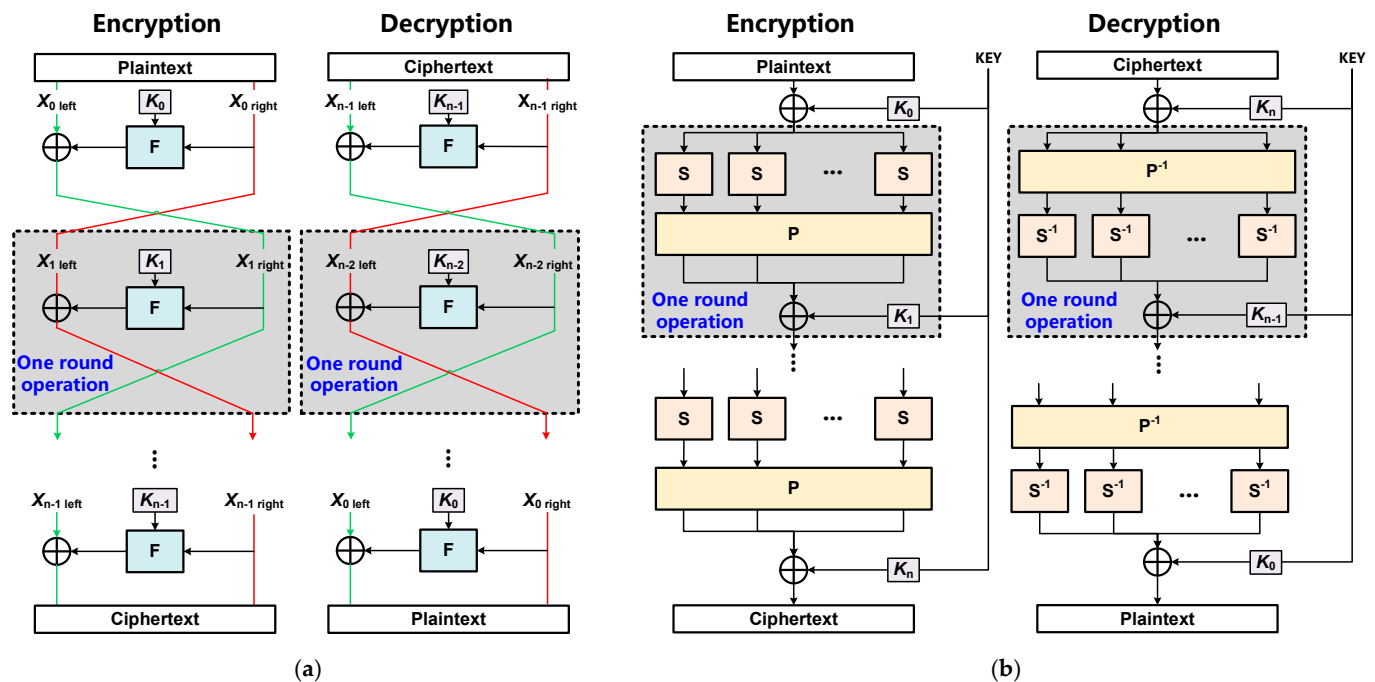


Figure 1. (a) Round operation structure of Feistel, (b) Round operation structure of SP.

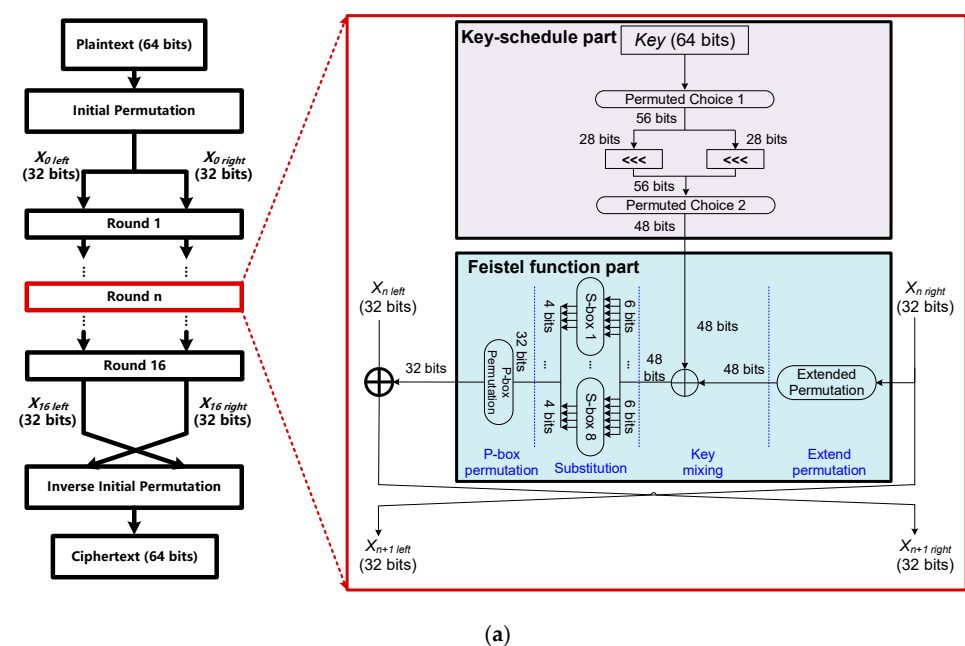


Figure 2. Cont.

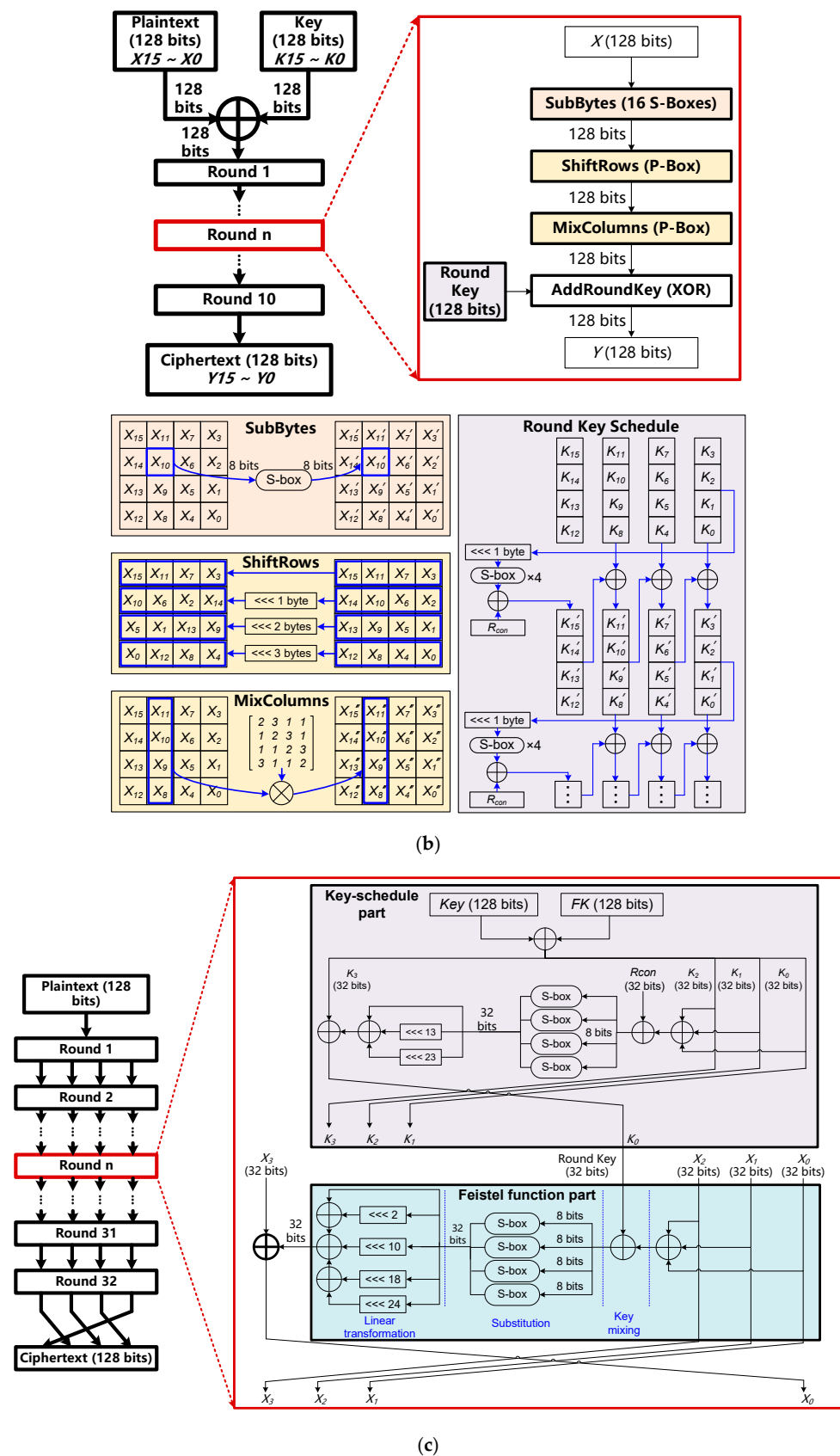


Figure 2. (a) The algorithm flowchart of DES based on the Feistel network, (b) The algorithm flowchart of AES based on the SP network, and (c) The algorithm flowchart of SM4 based on the Feistel network.

Table 2 summarizes the structure types and key operators of round operations in the three major block ciphers. It can be seen that both the DES and SM4 algorithms use the same round operation structure, i.e., Feistel structure, which means that the encryption and decryption data paths of DES/SM4 can be designed to a unified hardware. However, we can also find that, in fact there are more similar basic operators in the round operation of AES and SM4 algorithms, which can be exploited in order to reuse operators in the reconfigurable design to reduce hardware overhead.

Table 2. Structure types and key operators of round operations in the three major block ciphers.

Block Ciphers	Round Operation Structure Type	Key Operators in Round Operation
DES	Feistel	XOR/S-Box/Permutation
AES	SP	XOR/S-Box/Shift/Column Mix
SM4	Feistel	XOR/S-Box/Shift

Table 3 shows detailed information on key operators of round operations in the three major block ciphers. It can be seen that the key basic operators of SM4 are also included in the AES, while the majority of key basic operators in DES are different from the AES and SM4. It is worth noting that the same operators in different algorithms can be theoretically reused in hardware implementation. However, reusing operators also requires additional control hardware overhead, which reduces or even counteracts the advantage of reduced hardware overhead by reusing operators, especially for some simple logic operators or logic operators that are used less frequently. The 8-8 bit S-Box has the largest hardware overhead and the most times of usage, which is a key operator that needs to be reused. It is to be noted that for the AES and SM4 algorithms, because the mapping relationship between input and output of S-Box is inconsistent, there is no reusable part of the S-Box that is directly implemented based on look-up table. However, in the implementation of the S-Box based on Galois field, the calculation process of the two algorithms has the same inverse operation under GF (2^8) that will be introduced in detail in Section III, which can be exploited for reconfigurable design. In addition to the operators, reusing a large number of data iteration register files frequently used in round operations is also the key to building a unified data path.

Table 3. Detailed information on key operators of round operations in the three major block ciphers.

Key Operators	Input/Output Bit-Widths	Number of Operators Used in One Round Operation of Block Cipher		
		DES	AES	SM4
Permutation	64-56 bits	1	-	-
	56-48 bits	1	-	-
	32-48 bits	1	-	-
	32-32 bits	1	-	-
XOR	48 bits	1	-	-
	128 bit	-	1	1
	32 bit	1	4	14
S-Box	6-4 bit S-Box	8	-	-
	8-8 bit S-Box	-	20	8
Shift	28-bit circular left shift	2	-	-
	32-bit circular left shift	-	4	6
Column Mix (Inverse Column Mix)	32-bit (Inverse) column mix	-	4	-

From the above analysis, the key idea of the reconfigurable hardware design of block ciphers is to build a unified cipher data path by reusing the reconfigurable S-Boxes based on Galois field and data iteration register files.

2.2.2. Algorithm Analysis of Major Hash Functions

Hash functions including SHA-1/SHA-256/SM3 usually adopt the Merkle–Damgård structure [15], as shown in Figure 3. In the Merkle–Damgård diagram, the input message is firstly divided into several blocks of fixed size (512 bits in SHA-1/SHA-256/SM3) and then padded with bits representing the entire message length. The message blocks are processed with a series of cascaded one-way compression functions denoted by f , which transforms two inputs to an output with the same length as the initialization vector (IV), i.e., a fixed value for a specific hash function. For each one-way compression function f , the output should be operated with the input of the same function f by addition in SHA-1/SHA-256 or XOR in SM3, and the result of the operation is taken as the input of the next subsequent compression function f . The one-way compression function f of SHA-1/SHA-256/SM3 is shown in Figure 4. To complete a one-way compression function, the SHA-1, SHA-256 and SM3 need to perform 80, 64 and 64 rounds of operations, respectively. The final output message digest is obtained after all the message blocks with length padding have been compressed by the one-way compression functions.

Figure 4a shows the operation flow in the one-way compression function of SHA-1, with a 160-bit IV and a 160-bit output message digest [5]. For SHA-1, the 160-bit IV is first divided into 5×32 -bit values and cached by five buffers (A, B, C, D, E). The 512-bit message block is expanded into 80×32 -bit K_t corresponding to 80 round operations in each one-way compression function of the SHA-1. The 80 round operations are divided into four groups of 20 rounds and each group comprises the same circular left shift operations, the same addition modulo operations and different bitwise logical operations (F_t). The predefined values of K_t are also different in four different groups of round operations. Each round performs the calculation shown in Figure 4a and updates the five buffers (A, B, C, D, E).

Figure 4b shows the operation flow in the one-way compression function of SHA-256, with a 256-bit IV and a 256-bit output message digest [5]. Different from the SHA-1, the 256-bit IV is first divided into 8×32 -bit values and cached by eight buffers (A, B, C, D, E, F, G, H). The 512-bit message block is expanded into 64×32 -bit W_t corresponding to the 64 round operations in each one-way compression function of the SHA-256. Each round operation comprises additions and logical operations, which is more complex than the SHA-1. It is to be noted that the predefined values of K_t are different in each round operation, which is also more complicated than the SHA-1. Each round performs the calculation shown in Figure 4b and updates the eight buffers (A, B, C, D, E, F, G, H).

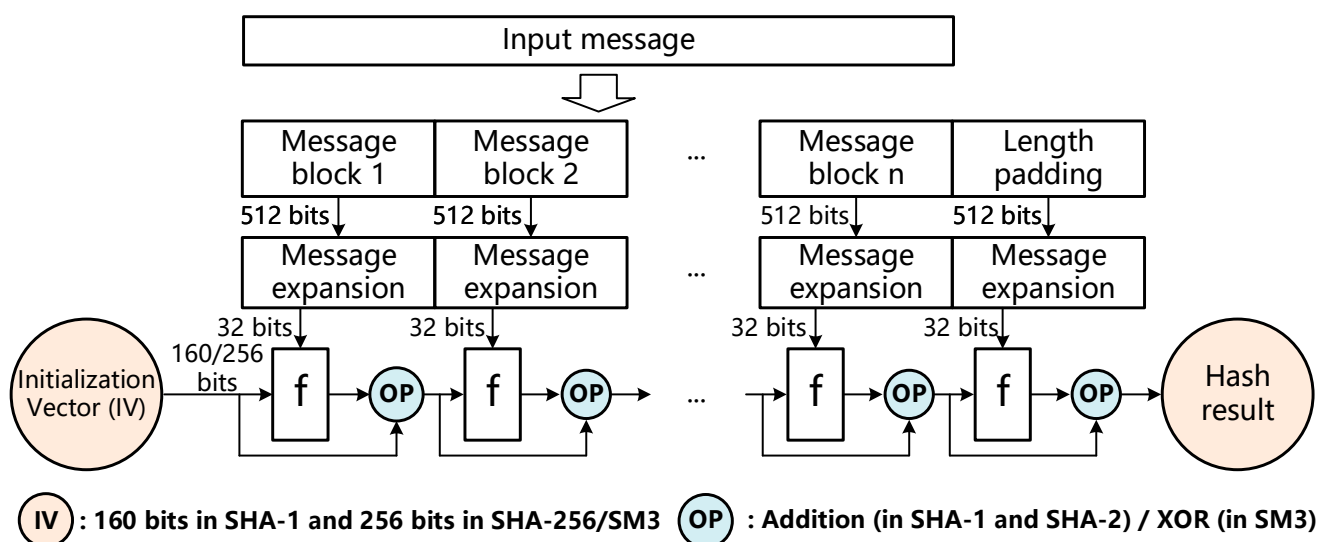
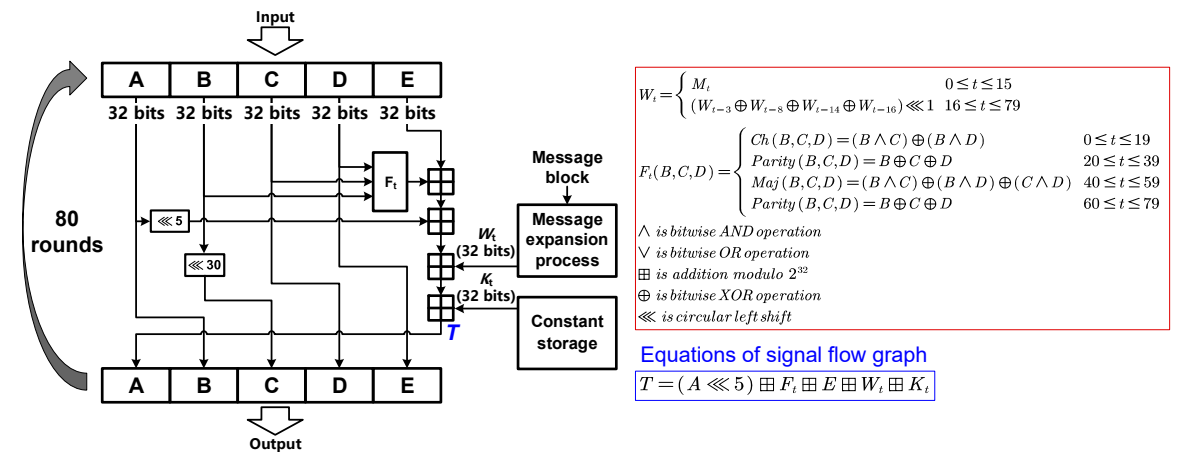
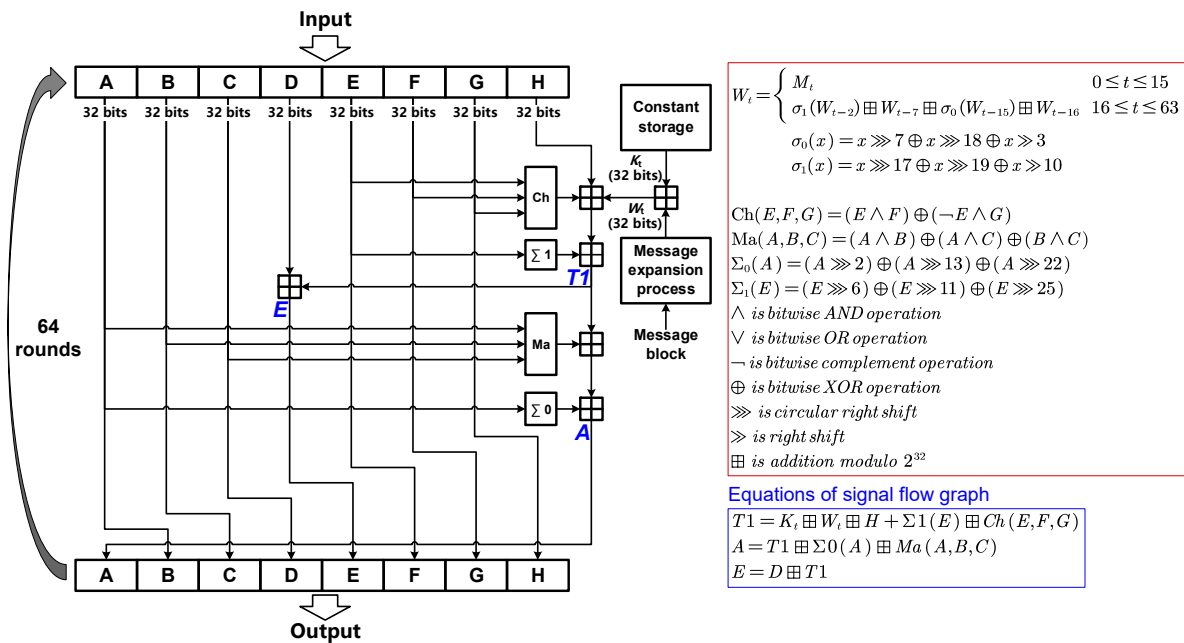


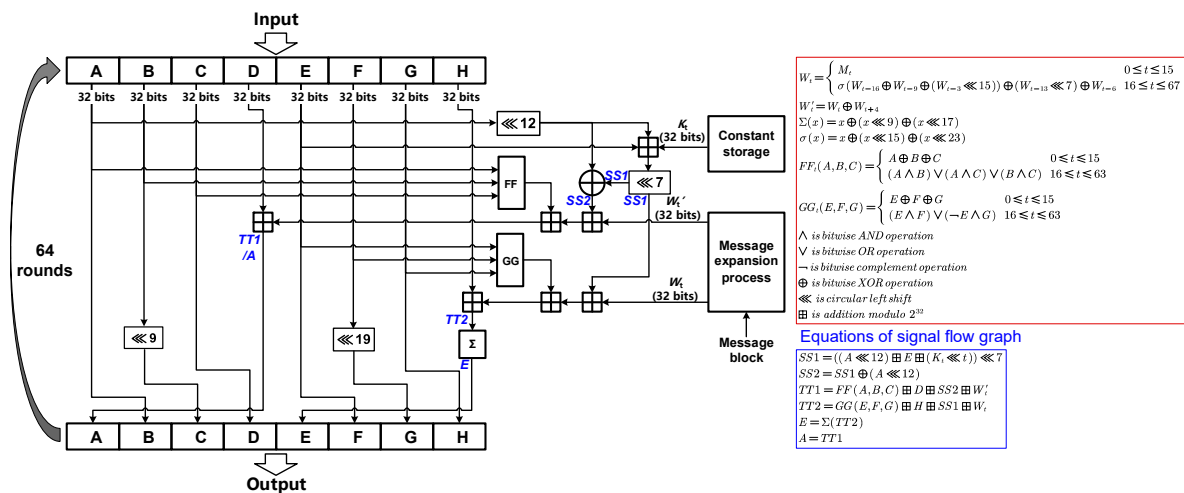
Figure 3. Merkle–Damgård construction of hash functions.



(a)



(b)



(c)

Figure 4. (a) The one-way compression function f of SHA-1, (b) of SHA-256, (c) and of SM3.

Figure 4c shows the operation flow in the one-way compression function of SM3, with a 256-bit IV and a 256-bit output message digest [6]. The division of 256-bit IV is the same as SHA-256. The 512-bit message block is expanded into 64×32 -bit W_t corresponding to the 64 round operations in each one-way compression function of the SM4. The logic operations of message expansion in the SM4 algorithm are different from those in the SHA-256. There are 64 round operations divided into two groups, one for the first 16 rounds and one for the last 48 rounds, respectively. Each group comprises the same circular left shifts, the same addition modulo operations and different bitwise logical operations (FF_t , GG_t). Similar to the SHA-256, each round performs the calculation and updates the eight buffers (A, B, C, D, E, F, G, H).

Table 4 summarizes the key operators of round operations in the three major hash functions. The key operators in SHA-1 and SM3 are the same, while the key operators in the SHA-256 round operation are partially different. It is worth noting that all three hash functions use the same Merkle–Damgård construction, which means that the preprocessing (message padding) process of the three hash algorithms is the same and that the corresponding hardware module can be reused. For efficient reconfigurable hardware design of the three hash algorithms, the reuse of modulo addition operator is the key due to its most frequent usage and the most complex operation. Like block ciphers, the reuse of iteration register files and message expansion register files is also the key to reducing hardware overhead and to building a unified data path.

Table 4. Detailed information of key operators in the three major hash functions.

Key Operators	Input and Output Bit-Widths	Number of Operators in One Round Operation of Hash Functions		
		SHA-1	SHA-256	SM3
XOR	32 bit	5	11	10
Shift	32-bit circular left shift	3	-	11
	32 bits circular right shift	-	10	-
	32 bits right shift	-	2	-
Modulo addition	32 bits ADD mod 2^{32}	4	9	7

From the above analysis, the key idea of the reconfigurable hardware design of hash functions is to reuse a preprocessing (message padding) module, build a unified compressed data path by reusing register files and modulo addition operators and develop a reconfigurable message expansion module by reusing register files.

2.2.3. Algorithm Comparison between Block Ciphers and Hash Functions

From the aforementioned algorithm analysis and discussion of block ciphers and hash functions, we can summarize three major differences between the two distinctive cryptographic primitives:

- The principle of the block ciphers and hash functions is fundamentally different. From the processing perspective, block ciphers work in two ways that are reversible, i.e., data encryption and decryption, while hash functions are irreversible that convert original data into message digest. From the application perspective, block ciphers are used to secure the data from the reach of third parties, while hash functions help protect the integrity of the information.
- The structure of round operations in the two cryptographic primitives is different. The round operation structure of the block cipher is based on Feistel or SP, while the round operation structure of hash functions is Merkle–Damgård based on one-way compression. The different structures of the two cryptographic primitives make the hardware data path quite different, which raises a great challenge to the development of a unified data path for efficient reconfigurable hardware design.

- The basic operators in the two cryptographic primitives are different. The key operator of block ciphers is an S-Box performing data substitution, while the key operators of hash functions are logical functions (i.e., F_t , Ch , Ma , FF_t , GG_t). The different basic operators actually impede the reconfigurable design of the two kinds of primitives.

From the hardware design perspective, the three aforementioned major differences make it inefficient to map the two primitives into a unified hardware accelerator. Specifically, a unified hardware accelerator implementing the two different kinds of primitives results in many redundant interconnections, including a large number of multiplexers (MUXs), routers and complex finite state machine logic, or even compilers to realize very complex control. Consequently, the approach of using unified hardware to implement two different kinds of primitives has high hardware complexity, large hardware overhead, and high power consumption.

Furthermore, as mentioned previously, considering many application scenarios of different standards and protocols, it is necessary to perform decryption and hash authentication simultaneously when receiving the ciphertext and message digest at the same time. The approach of using unified hardware requires an operation manner of time division multiplexing, i.e., reconfiguring the hardware into one of either the hash authentication and block decryption mode after the other mode completes its processing task. In contrast, two separate reconfigurable hardware units for these two kinds of primitives can perform decryption and hash authentication at the same time, which improves computing speed and efficiency. Therefore, in this study, two separate reconfigurable cryptographic units are designed to implement block ciphers and hash functions, respectively.

3. Hardware Architecture of Proposed Reconfigurable Cryptographic Accelerator

Based on the aforementioned analysis of DES/AES/SM4 and SHA-1/SHA-256/SM3 algorithms and the characteristics of operators, a hardware architecture of reconfigurable cryptographic accelerator is proposed, which is suitable for resource-limited and energy-constrained IoT devices including small mobile intelligent robots, wearable medical devices and miniature unmanned aerial vehicles. Specifically, the proposed reconfigurable cryptographic accelerator can be integrated as a dedicated cryptographic engine into System-on-Chip (SoC) chips for robot mapping, positioning and navigation, wearable medical monitoring devices and miniature unmanned aerial vehicles. The accelerator consists of a reconfigurable cipher unit and a reconfigurable hash unit, which are integrated with a unified advanced extensible interface (AXI) as shown in Figure 5.

3.1. Reconfigurable Cipher Unit

Figure 5 presents the proposed reconfigurable cipher unit that can perform the encryption and decryption operations of three kinds of block ciphers including DES, AES-128/AES-192/AES-256 and SM4 algorithms. The unit mainly includes the encryption/decryption round operation part, key-schedule part and round control logic, which mainly configures different block ciphers by counting different rounds using a counter to control the reconfigurable cipher unit.

For the encryption/decryption round operation part, the data iteration register files ($X_0, X_1, \dots, X_{15}$) in light blue color are reused for the DES, AES and SM4 algorithms, and the reconfigurable S-Boxes in light blue color are reused for AES and SM4 algorithms. The three operation modes of this part are described as following:

- In the DES encryption/decryption mode using the logical operators highlighted by red boxes, eight data iteration register files (X_0 – X_7) are used to store the 64-bit plaintext. The data path is configured to use the extended permutation module, 48-bit XOR logic, eight S-Boxes, P-Box permutation module and the final 32-bit XOR logic, which are all dedicated to DES due to the distinct difference of operators between DES and AES/SM4.
- In the AES encryption mode using the logical operators highlighted by green boxes, sixteen data iteration register files (X_0 – X_{15}) are used to store the 128-bit plaintext. The

data path is configured to use the sixteen reconfigurable S-Boxes, circular shift module, MixColumn module and the final 128-bit XOR logic to realize SubBytes, ShiftRows, MixColumns and AddRoundKey, respectively. In AES decryption mode, there are two differences from the AES encryption mode. First, the mode of the sixteen reconfigurable S-Boxes and circular shift module are changed to the AES decryption mode. Second, the MixColumn module is replaced by the InvMixColumn module according to the AES standard. Note that the modules, including circular shift, MixColumn and InvMixColumn, are dedicated to the AES mode as there are no such operations in the other two block ciphers.

- In the SM4 encryption/decryption mode using the logical operators highlighted by blue boxes, and as with the AES, sixteen data iteration register files (X_0 – X_{15}) in light blue color are reused to store the 128-bit plaintext. The data path is configured to use the 32-bit XOR logic, four reused reconfigurable S-Boxes, linear transformation module and the final 32-bit XOR logic. Note that the linear transformation module is dedicated to the SM4 mode.

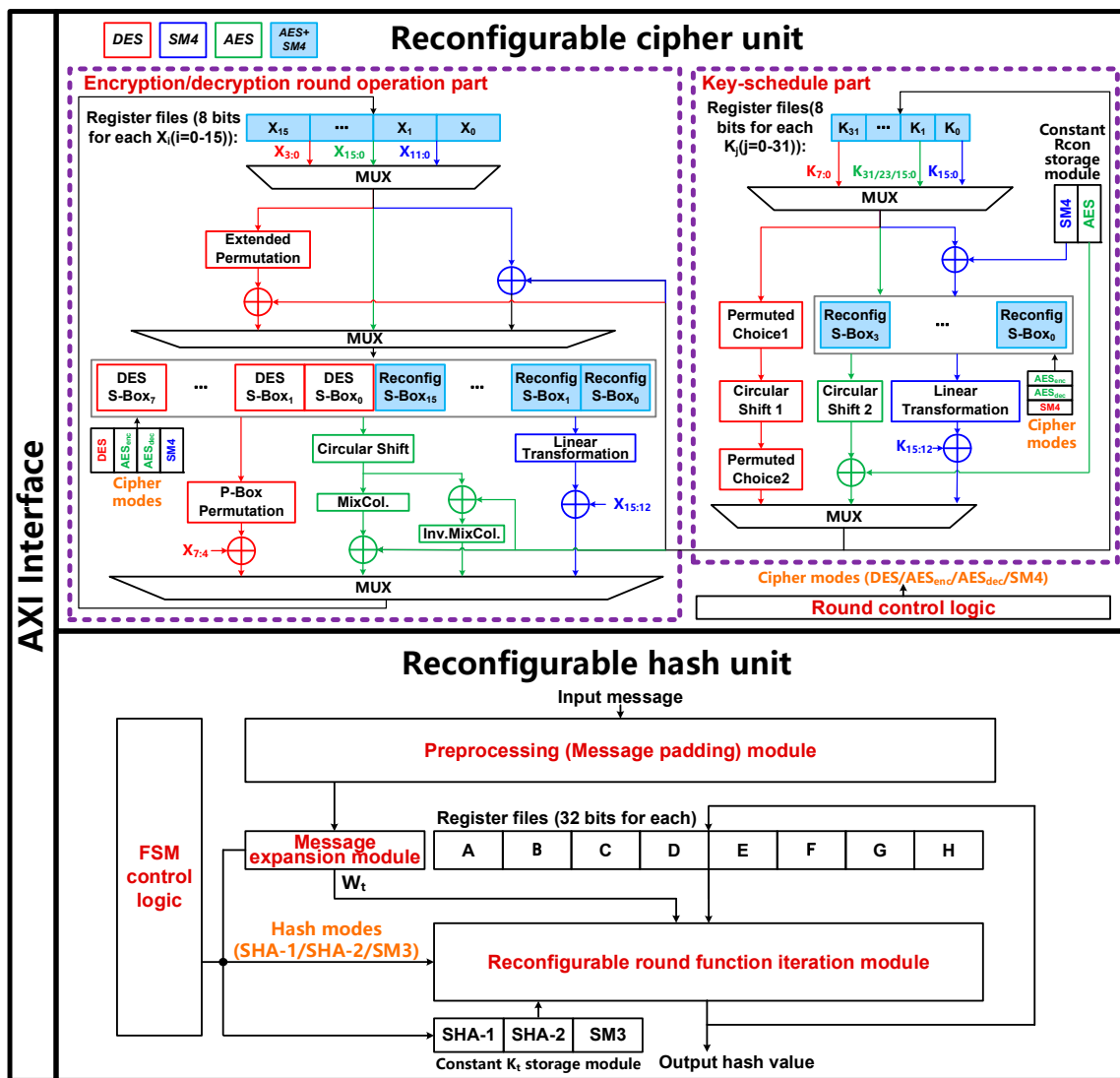


Figure 5. The proposed reconfigurable cryptographic accelerator with reconfigurable cipher unit and reconfigurable hash unit integrated with a unified AXI interface.

For the key-schedule part, the round key register files ($K_0, K_1, \dots, K_{31}$) are reused for the DES, AES and SM4 algorithms, and reconfigurable S-Boxes are also reused for AES

and SM4 algorithms. The mode analysis is almost the same as the encryption/decryption round operation part shown in Figure 5. It is worth noting that there are 16/24/32 round key register files used for AES-128/192/256 algorithm and four reconfigurable S-Boxes reused for AES-128/192/256 and SM4.

For the design of S-Box, a unified on-the-fly reconfigurable S-Box based on Galois fields (i.e., $GF(2^8)$) is designed for the encryption and decryption of AES and SM4 to address the problem of large hardware overhead and non-reusability of S-Box based on LUT implementation. The reconfigurable S-Box module for AES and SM4 is shown in Figure 6. There are 8-bit input data and 8-bit output data for the three operation modes including the AES encryption mode, AES decryption mode and SM4 mode. Among the $GF(2^8)$ operations, only the inverse operation of composite field $GF(((2^2)^2)^2)$ module in light blue color can be reused for the three modes.

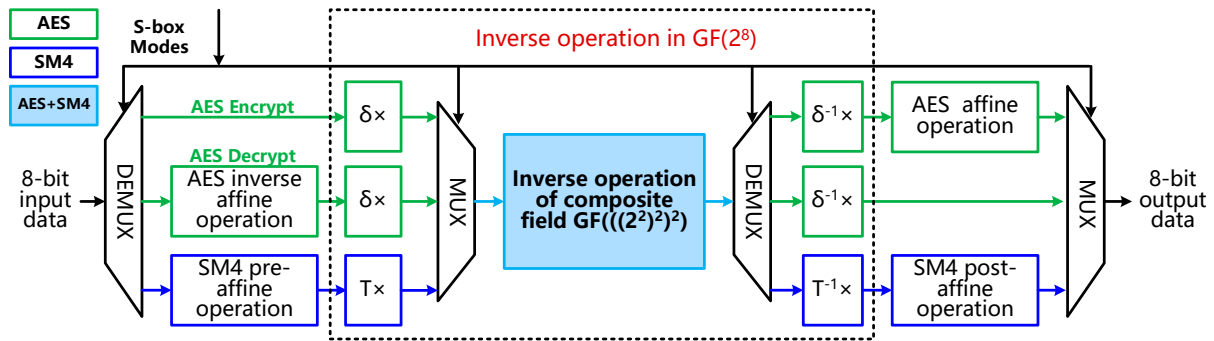


Figure 6. The reconfigurable S-Box module for AES and SM4.

The main idea of designing the reconfigurable S-Box is to reuse the major operation, i.e., the multiplicative inverse operation of AES and SM4 from the analysis of the operations of the S-Boxes in $GF(2^8)$. We have found that it is necessary to transform the multiplicative inverse operation from $GF(2^8)$ to the $GF(((2^2)^2)^2)$ composite field because the hardware complexity of the operation in $GF(2^8)$ is too high. The transformation can be done by the calculation of the isomorphic mapping matrix. Finally, the multiplicative inverse operation in the $GF(2^8)$ field can be realized by bit-wise logic operations in $GF(2^2)$ and, therefore, the corresponding efficient hardware architecture of the reconfigurable S-Box can be obtained. The hardware implementation details are described as below.

The S-Box operation of AES encryption includes multiplicative inverse operation in the $GF(2^8)$ field and affine operation, while the S-Box operation of AES decryption includes inverse affine operation and multiplicative inverse operation in the $GF(2^8)$ field. The S-Box operation of SM4 encryption/decryption includes pre-affine operation, multiplicative inverse operation in the $GF(2^8)$ field and post-affine operation. The corresponding operation in the $GF(2^8)$ field is shown in Equations (1) and (2) [16].

$$Z = M(X)^{-1} + V \quad (1)$$

$$Y = A(AX + C)^{-1} + C \quad (2)$$

where X is the input of the S-Box, Z and Y respectively represent the output of the AES and SM4 S-Box, M and V represent the affine matrix of the AES S-Box and the constants in the affine operation, respectively, and A and C represent the affine matrix of the SM4 S-Box and the constants in the affine operation, respectively. The irreducible polynomial ($f(x)_{AES}$ and $f(x)_{SM4}$), corresponding to the multiplicative inverse operation $(X)^{-1}$ and $(AX + C)^{-1}$ in the finite field $GF(2^8)$ of AES and SM4 are shown in Equations (3) and (4), respectively.

$$f(x)_{AES} = x^8 + x^4 + x^3 + x^1 + 1 \quad (3)$$

$$f(x)_{SM4} = x^8 + x^7 + x^6 + x^5 + x^4 + x^2 + 1 \quad (4)$$

From Equations (1) and (2), it can be observed that the multiplicative inverse operation $(X)^{-1}$ and $(AX + C)^{-1}$ in the finite field $GF(2^8)$ are the main components and similar operations of the two S-Boxes, which can be designed with the reconfigurable method to reduce the hardware overhead. However, the implementation of multiplicative inverse operation in the $GF(2^8)$ field requires the extended Euclid algorithm, which is computationally intensive and not suitable for hardware implementation [17,18]. According to the properties of the finite field, the operation in $GF(2^8)$ can be converted to the composite field $GF(((2^2)^2)^2)$ through isomorphic transformation, so as to reduce the amount of computation. The isomorphic mapping from the finite field $GF(2^8)$ to the composite field $GF(((2^2)^2)^2)$ is represented by an 8×8 matrix (matrix δ for AES and matrix T for SM4) [17]. The 8×8 isomorphic mapping matrix δ and T for AES and SM4 used in this work is shown as follows:

$$\delta = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}, \quad T = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (5)$$

Assume the inverse of $g = (a_1 Y^{16} + a_0 Y)$ is $h = (d_1 Y^{16} + d_0 Y)$, $a_1, a_0, d_1, d_0 \in GF(2^4)$, $[Y^{16}, Y]$ is a set of regular bases in the $GF(2^4)$ field, and is two roots of irreducible polynomial, i.e., $r(y) = y^2 + y + \eta = 0$ (η is a constant and is specified for each algorithm). Then the h can be calculated as follows:

$$h = (d_1 Y^{16} + d_0 Y) = (\theta^{-1} a_0) Y^{16} + (\theta^{-1} a_1) Y, \quad \theta = (a_1 a_0 + (a_1^2 + a_0^2) \eta) \quad (6)$$

in which way the multiplicative inverse operation in $GF(2^8)$ can be transformed into operations in $GF(2^4)$ including multiplication, inverse operation and other operations. Similarly, the inverse of $a = (b_1 Z^4 + b_0 Z)$, i.e., $i = (e_1 Z^4 + e_0 Z)$, $b_1, b_0, e_1, e_0 \in GF(2^2)$, $[Z^4, Z]$ is a set of regular bases in the $GF(2^2)$ field, and is two roots of irreducible polynomial, i.e., $t(z) = z^2 + z + \rho = 0$ (ρ is a constant and is specified for each algorithm). Then i can be calculated as:

$$i = (e_1 Z^4 + e_0 Z) = (\tau^{-1} b_0) Z^4 + (\tau^{-1} b_1) Z, \quad \tau = (b_1 b_0 + (b_1^2 + b_0^2) \rho) \quad (7)$$

The operation structure diagram corresponding to Equations (6) and (7) is shown in Figures 7 and 8, respectively.

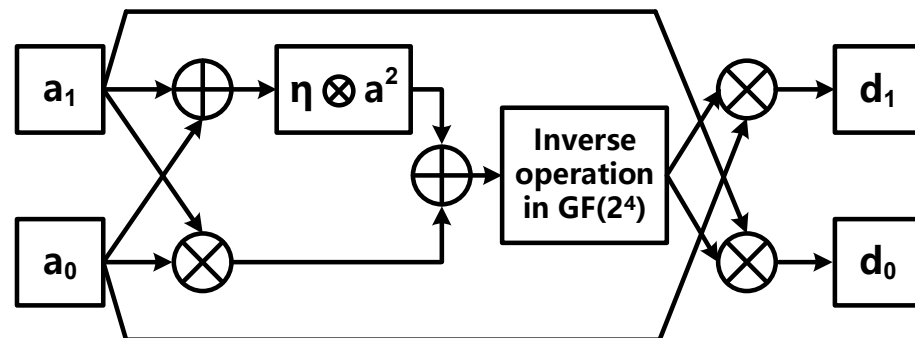


Figure 7. Multiplicative inverse operation of $GF(2^8)$ based on regular basis.

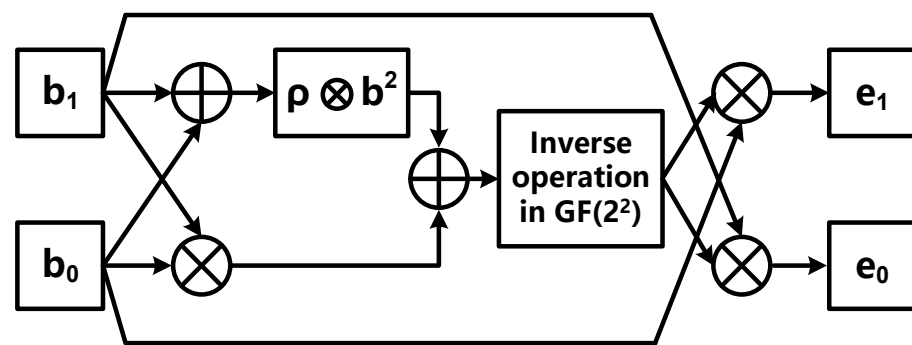


Figure 8. Multiplicative inverse operation of $GF(2^4)$ based on regular basis.

In $GF(2^2)$, the multiplicative inverse operation is the same as the square operation. When regular basis is used, the multiplicative inverse operation can be further calculated as:

$$(c_1W^2 + c_0W)^{-1} = (c_1W^2 + c_0W)^2 = (c_0W^2 + c_1W) \quad (8)$$

in which way the multiplicative inverse operation in the $GF(2^2)$ field can be easily implemented in bit-wise logic operations as mentioned previously.

Table 5 shows the hardware overhead of S-Boxes based on LUT and GF in a 65-nm ASIC implementation. It can be found that the S-Box based on $GF(((2^2)^2)^2)$ field implementation can save hardware overhead by more than 30% against the LUT implementation.

Table 5. Hardware overhead of S-Boxes based on LUT and $GF(((2^2)^2)^2)$ in 65-nm ASIC implementation.

Algorithm Type	Implementation	Gate Count	Saving
AES	LUT	586	N/A
	GF	335	42.83%
SM4	LUT	368	N/A
	GF	244	33.70%

Table 6 shows the comparison of hardware overhead between the reconfigurable cipher unit and the separate hardware implementation of the DES/AES/SM4 in 65 nm, where the gate count is reduced by 33.65%. There are two main reasons for the reduction of hardware overhead. First, the reuse of iteration register files in the unified encryption/decryption and key-schedule data paths allows the number of iteration register files to be reduced by more than 50%. Second, by designing the reconfigurable S-Box based on the composite field $GF(((2^2)^2)^2)$, the gate count of the S-Box can be reduced by about 50% as compared to the conventional LUT implementation method. At the same time, the reuse of S-Boxes by AES and SM4 further achieves a reduction of eight S-Boxes as compared to the separate implementation scheme.

Table 6. Hardware overhead of separate hardware implementation and reconfigurable implementation of the three block ciphers in 65 nm.

Structure Type	Algorithm Type	Gate Count	Saving
Separate	DES-64	1956	N/A
	AES-128/192/256	19,518	N/A
	SM4-128	5635	N/A
Total Reconfigurable	DES+AES+SM4	27,109	N/A
	DES/AES/SM4	17,986	33.65%

3.2. Reconfigurable Hash Unit

As shown in Figure 5, the reconfigurable hash unit can perform three kinds of hash functions including SHA-1, SHA-256 and SM3, which mainly comprises six sub-blocks:

- The preprocessing module mainly performs the original message padding to assure that the input data block is a multiple of 512 bits.
- The message expansion module expands each 512-bit input data block into a predefined round number of words, i.e., round number $\times W_t$, which is 80×32 bits in SHA-1 and 64×32 bits in SHA-256/SM3.
- The reconfigurable round function iteration module with a unified compressed data path performs 80/64/64 round operations to achieve the one-way compression function f of SHA-1/SHA-256/SM3 for one 512-bit input data block. As a result, the final hash value is obtained after all the 512-bit data blocks have been computed.
- The constant K_t storage module stores the constant K_t used for each round operation.
- The iteration register files (A, B, ..., H) store the initial IV in the first round operation and intermediate value obtained from each round operation.
- The finite state machine (FSM) control logic controls the reconfigurable hash unit mainly by generating control signals from counters to control MUXes for realizing reconfigurable computation of the three algorithms.

As mentioned before, the key idea of the reconfigurable hardware design of hash functions is to reuse the preprocessing module, develop the reconfigurable message expansion module by reusing register files, and build a reconfigurable round function iteration module with a unified compressed data path by reusing register files and modulo addition operators.

First, the preprocessing module can directly be reused by SHA-1, SHA-256 and SM3 because the preprocessing procedure is the same as for transforming the input n -bit message into a multiple of 512-bit data blocks by padding for the three hash functions. The padding procedure performs padding on the original n -bit input message by adding one bit of "1", k bits of "0", and 64 bits of the original message size, by following the equation of $(n + 1 + k = 448 \bmod 512)$ to find the minimum k value.

Second, the hardware architecture of the reconfigurable message expansion module is designed to reuse the shift register files (i.e., $W_0, W_1, \dots, W_{15}$) that are used to store the initial input message of the 512-bit data block from the preprocessing module, and shift the expanded messages during round operations, as shown in Figure 9. For the SM3 mode, the data in the W_{13}, W_{10}, W_7, W_3 and W_0 are taken out at each clock cycle to perform circular shift and XOR operation. The σ operator is described by a function containing circular shift and XOR operations in Figure 4c. The output W_t is described by the following Equations (9) and (10):

$$W_t = \sigma(W_{t-16} \oplus W_{t-9} \oplus (W_{t-3} \lll 15)) \oplus (W_{t-13} \lll 7) \oplus W_{t-6} \quad (9)$$

$$W'_t = W_t \oplus W_{t+4} \quad (10)$$

For the SHA-1 mode, the data in the W_{13}, W_8, W_2 and W_0 are taken out at each clock cycle to perform circular shift and XOR operation as described by the following Equation (11):

$$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1 \quad (11)$$

Because the input number of XOR in SHA-1 is different from that of SM3, the XOR operator is not reused for the two operation modes. For the SHA-256 mode, the data in the W_{14}, W_9, W_1 and W_0 are taken out at each clock cycle to perform the circular shift and XOR operation contained in the functions of σ_0 and σ_1 . Different from the first two modes of SM3 and SHA-1, the SHA-256 mode has the addition modulo 2^{32} operation as shown in Figure 4b and Equation (12):

$$W_t = \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16} \quad (12)$$

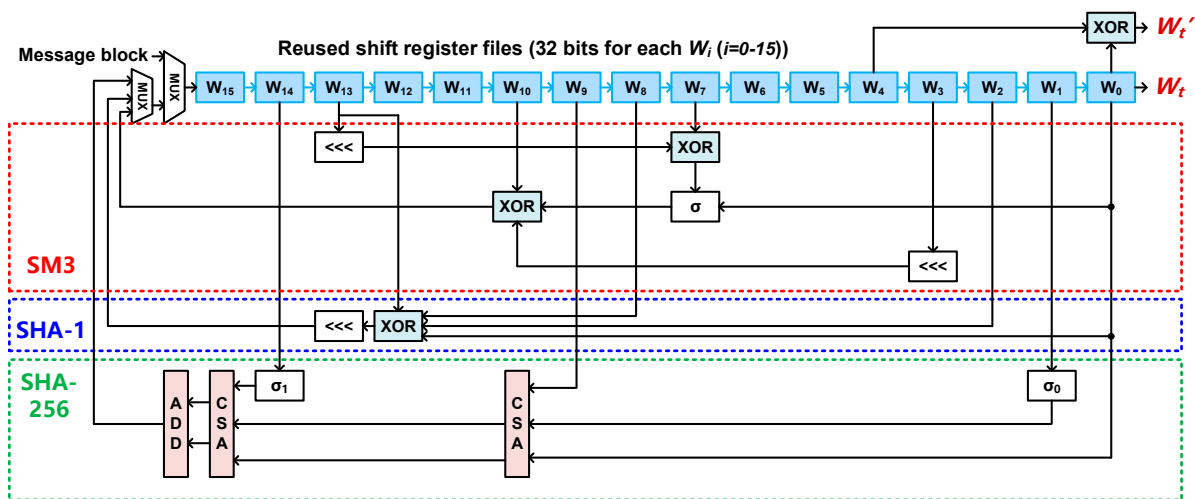


Figure 9. Hardware architecture of message expansion module in the reconfigurable hash unit.

Therefore, it is difficult to reuse the operators for message expansion in the three modes for a reconfigurable design. For the three modes, the only shared hardware resources are shift registers, i.e., W_{15} to W_0 . The expanded value W_t is processed by the logic operations for 80 (in SHA-1) or 64 rounds (in SHA-256 and SM3). For each round, the logic expansion result of W_t is feedback to the input of the register file W_{15} .

Third, the reconfigurable round function iteration module is designed to build a unified compression data path by reusing register files and modulo addition (sum and modulo 2^{32}) operators in light blue color as shown in Figure 10a. Based on the observation that operations in the one-way compression of SM3 contains most operations in both SHA-1 and SHA-256, the main idea is to first design the SM3 data path consisting of part 1, part 2 and part 3 as shown in Figure 10a, and then add a small amount of arithmetic logic and MUXes appropriately to achieve a reconfigurable design supporting the other two hash functions. The module performs multiple rounds of compression operation from the initialization vector (IV) with the expanded value W_t and the constant K_t to calculate the final hash result. The three operation modes of the reconfigurable design are described as following:

- For the SM3 mode, the hardware data path is implemented by direct mapping from the signal flow data path shown in Figure 4d. Specifically, part 1 calculates the intermediate value $SS1$ and $SS2$ by using constant K_t and values in input register files A and E. At the same time, part 2 calculates the intermediate value $TT2$ by using the expanded value W_t , the $SS1$ from part 1, and values in input register files E, F, G and H. Part 3 calculates the value $TT1$ by using $SS2$ from part 1, the expanded value W_t' and values in input register files A, B, C and D.
- For the SHA-256 mode, three carry save adders (CSAs) in the light blue color are reused to calculate the intermediate value $T1$ in part 2. Another CSA and one ADD in light blue color are reused to calculate the value in output register file E in part 2. Two extra CSAs and one ADD in light blue color are also reused to calculate the value in output register file A in part 3. In summary, there are six CSAs and two ADDs reused in the SHA-256 mode. The other logic operators are designed to be dedicated to SHA-256 as they are difficult to be reused for reconfigurable design.
- For the SHA-1 mode, similar to the SHA-256 mode, three CSAs in light blue color are reused to calculate the intermediate value T in part 2. Two extra CSAs and one ADD in light blue color are reused to calculate the value in output register file A in part 3. In summary, there are five CSAs and one ADD reused in the SHA-1 mode. The other logic operators are also designed to be dedicated to SHA-1 as they are difficult to be reused for reconfigurable design.

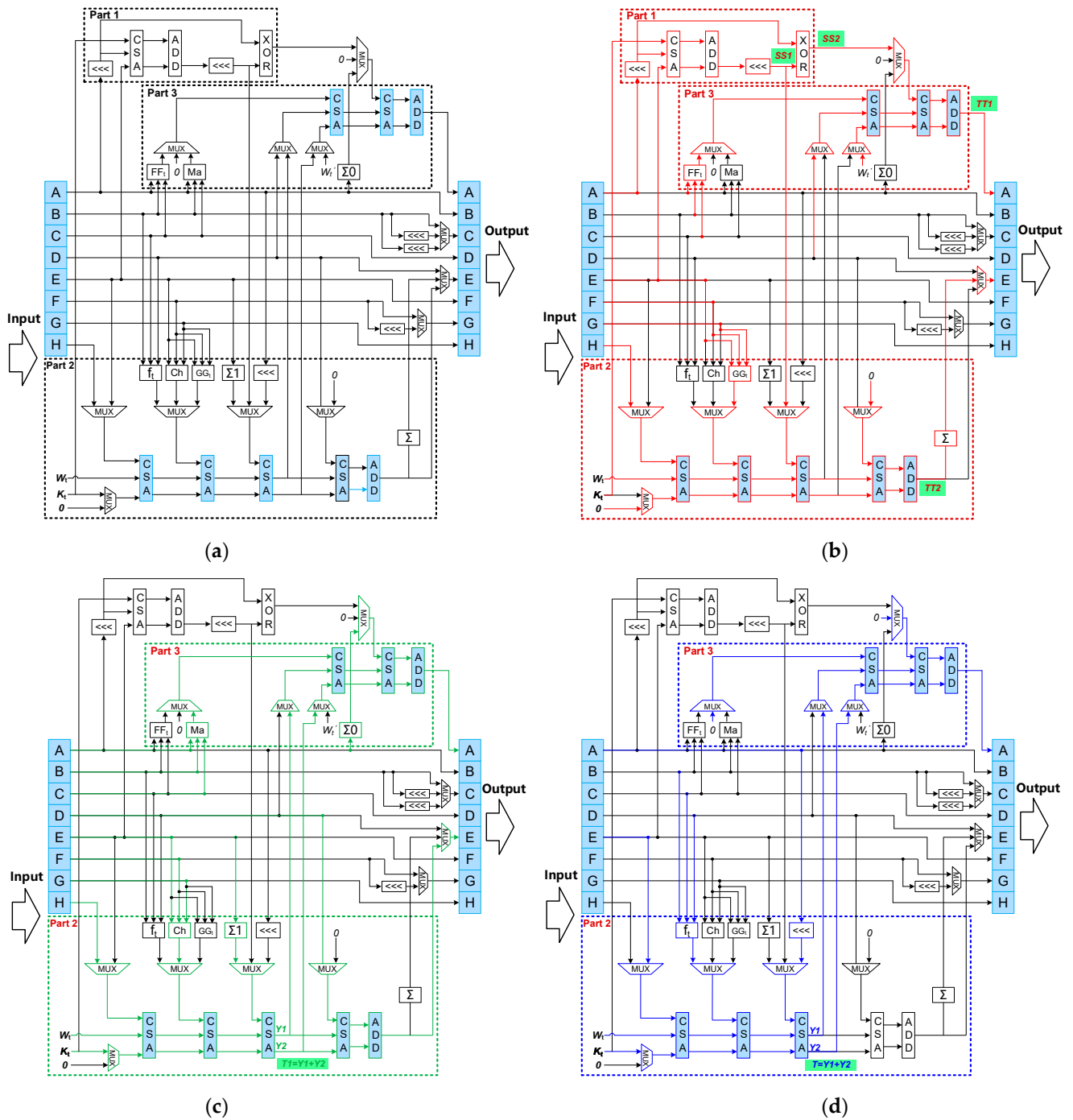


Figure 10. (a) Hardware architecture of the proposed reconfigurable round function iteration module in the reconfigurable hash unit, and its three operation modes for (b) SM3, (c) SHA-256 and (d) SHA-1, respectively.

Table 7 shows the comparison of hardware overhead between the reconfigurable hash unit and the separate hardware implementation of the SHA-1/SHA-256/SM3 in 65 nm, where the gate count reduces by 56.18%. There are three main reasons for the reduction of hardware overhead. First, only one preprocessing module is used in the reconfigurable hash unit, while three preprocessing modules are required in the separate implementation scheme. Second, in the message expansion module, sixteen 32-bit register files are reused. Third, in the unified one-way compression data path, registers and addition modulo operators are reused to further reduce the hardware overhead.

Table 7. Hardware overhead of separate hardware implementation and reconfigurable implementation of the three hash functions in 65 nm.

Structure Type	Algorithm Type	Gate Count	Saving
Separate	SHA-1	16,055	N/A
	SHA-256	18,627	N/A
	SM3	17,921	N/A
Total	SHA-1+SHA-256+SM3	52,603	N/A
Reconfigurable	SHA-1/SHA-256/SM3	23,050	56.18%

4. Implementation Results and Discussions

This section presents the results of the proposed reconfigurable cryptographic accelerator based on both Xilinx Virtex UltraScale + Field programmable gate array (FPGA) and 65-nm application-specific integrated circuit (ASIC) implementations. The hardware overhead, speed performance and energy efficiency are presented and discussed comprehensively. The comparison with the existing reconfigurable designs is also performed and discussed.

4.1. FPGA Implementation and Evaluation

The separate hardware implementation and reconfigurable implementation of three block ciphers and hash functions are realized based on a Xilinx Virtex UltraScale+ FPGA device. Table 8 shows the hardware overhead of separate hardware implementation and reconfigurable implementation.

Table 8. Hardware overhead of the separate hardware implementation and the reconfigurable implementation of the three block ciphers and hash functions based on FPGA.

Structure Type	Algorithm Type	LUTs	Registers	F7 MUXes	F8 MUXes
Separate	DES-64	342	74	96	0
	AES-128/192/256	3756	685	649	256
	SM4-128	732	266	103	20
Total	DES+AES+SM4	4830	1025	848	276
Reconfigurable	DES/AES/SM4	5555	849	83	0
Separate	SHA-1	1980	2910	0	0
	SHA-256	2549	2922	0	0
	SM3	2043	2939	0	0
Total	SHA-1+SHA-256+SM3	6572	8771	0	0
Reconfigurable	SHA-1/SHA-256/SM3	3140	2951	0	0

For the implementation of three block ciphers, although the LUTs of the reconfigurable implementation increase by 15.0% compared with the separate hardware implementation, the registers, F7 MUXes and F8 MUXes, of the reconfigurable implementation reduce by 17.2%, 90.2% and 100%, respectively, as compared to the separate implementation. There are two main reasons for the reduction of hardware overhead. First, the number of iteration register files are reduced by more than 50% through the reuse of register files. Second, the hardware overhead of the S-Box based on the composite field $GF(((2^2)^2)^2)$ can be reduced by about 30% as compared to the LUT implementation. At the same time, the reuse of S-Boxes by AES and SM4 further achieves a reduction of eight S-Boxes. For the implementation of three hash functions, the LUTs and registers of the reconfigurable implementation reduce by 52.2% and 66.4%, respectively. The implementation results based on FPGA are consistent with the results of ASIC implementation shown in Tables 6 and 7, both of which show that reconfigurable hardware design has achieved more than 30%

and 50% reduction in hardware resource overhead for block ciphers and hash functions, respectively. There are three main reasons for the reduction of hardware overhead. First, only one preprocessing module is used in the reconfigurable hash unit. Second, in the message expansion module, sixteen 32-bit register files are reused. Third, in the unified one-way compression data path, registers and addition modulo operators are reused to further reduce the hardware overhead.

4.2. ASIC Implementation and Discussion

The proposed reconfigurable cryptographic accelerator core has been implemented into a 65-nm complementary metal oxide semiconductor (CMOS) process technology by a digital ASIC design flow based on Cadence Innovus Place and Route design tool. Figure 11 shows the ASIC layout with a core size of $286.2\ \mu\text{m} \times 581\ \mu\text{m}$, where the left part is the reconfigurable cipher unit and the right part is the reconfigurable hash unit.

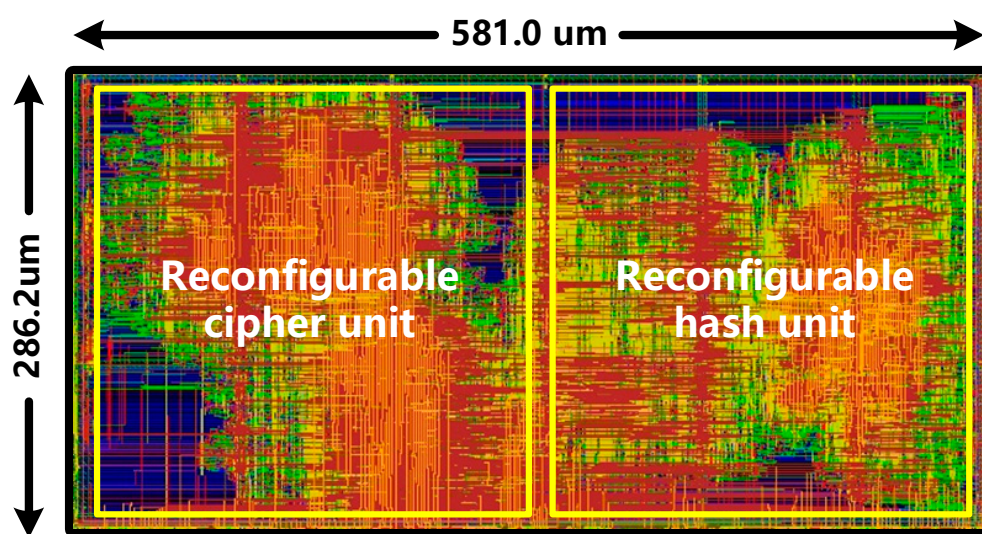


Figure 11. ASIC layout of the proposed reconfigurable cryptographic accelerator core in 65-nm CMOS process.

Table 9 shows the overall comparison between the proposed reconfigurable cryptographic accelerator and the state-of-the-art designs based on ASIC implementation. The benchmark table uses AES-128 algorithm for a fair comparison, which has been widely adopted in many studies [10–12].

As shown in Table 9, the proposed reconfigurable cryptographic accelerator achieves the minimum area, the lowest power consumption, and the highest energy efficiency and area efficiency—mainly due to the different implementation methods of various cryptographic algorithms (DES/AES/SM4 and SHA-1/SHA-256/SM3). The implementation method in [10–12] is designing a unified reconfigurable engine based on the coarse-grained reconfigurable array (CGRA), which can realize a more flexible reconfigurable architecture supporting various cryptographic algorithms including block ciphers, stream ciphers and hash functions, etc. Specifically, in the three designs, memories are used to store configuration information, FIFOs are used to buffer intermediate data, and reconfigurable processing element arrays including a large number of reconfigurable cells, multiplexers and routers are used to realize a flexible reconfigurable architecture. The works in [10–12] have realized a more flexible reconfiguration to support much more cryptographic algorithms, while sacrificing area and energy efficiency, mainly due to the complex routing, interconnections and control logic as compared to our proposed work. On the other citations power consumption by about 49% and 18%, respectively. The significant reduction mainly benefits from the proposed design methods. These methods entail the building of a unified cipher data path through the reuse of reconfigurable S-Boxes based on Galois field and data iteration

register files in the reconfigurable hardware design of block ciphers. They also entail the reuse of a preprocessing (message padding) module, the building of a unified compressed data path, and the development of a reconfigurable message expansion module in the reconfigurable hardware design of hash functions.

Table 9. Overall comparison of the proposed reconfigurable cryptographic accelerator and existing designs based on ASIC implementation.

	TCAD'2018 [10]	TCAS-II'2020 [11]	TVLSI'2020 [12]	Ours Proposed Design
Major key algorithms	DES/AES/SM4 ZUC/SNOW/RC4 SHA-256/SM3	DES/AES/SM4 ZUC/SNOW/RC4 SHA-256/SM3	DES/AES/SM4 ZUC/RC4 SHA-256	DES/AES/SM4 SHA-1/SHA-256/SM3
Architecture	Reconfigurable architecture based on one unified data path	Reconfigurable architecture based on one unified data path	Reconfigurable architecture based on one unified data path	Reconfigurable architecture based on two unified data paths for cipher block and hash function algorithms
Technology	65 nm	65 nm	55 nm	65 nm
Area (mm ²)	7.75	9.91	12.25	0.11
Gate count	1910 K	N/A	N/A	41 K
Memory	97.9	N/A	N/A	0
Frequency (MHz)	400	500	110	323
Power (mW)	620 (155 *)	625 (125 *)	35 (31.81 *)	23 (7.12) *
Throughput (Gbps)	51.2 (12.8 *)	64 (12.8 *)	0.44 (0.4 *)	4.13 (1.28) *
Energy efficiency (Gbps/W)	82.6	102.4	15.71	441
Area efficiency (Gbps/mm ²)	6.61 (1.65 *)	6.46 (1.29 *)	0.04 (0.03 *)	37.55 (11.64) *
S-Box topology	LUT	LUT	LUT	GF(((2 ²) ²) ²)

*: scaled to 100 MHz.

For a case study of energy efficiency improvement for the proposed reconfigurable cryptographic accelerator, a typical lithium coin battery with 37 mAh is selected [19]. For a 3-lead 360-Hz 12-bit electrocardiogram (ECG) recording system, there are about 131 MBytes of raw data for one day, which needs to be processed by AES-128 and SHA-256. The 65-nm ECG SoC in [20] is selected to integrate the cryptographic accelerator. For a smart sensor node, usually 20% of battery power [21] can be allocated by the ECG SoC, which consumes 0.254 J per day when the cryptographic accelerator is not integrated. The 65-nm design in [11] is selected for comparison, which has the highest energy efficiency among the three designs in [10–12]. For the AES-128 cipher block processing of the 131 MBytes of raw data, the energy consumption per day is 0.00325 J and 0.01 J, by the design in [11] and our proposed design, respectively. For the SHA-256 hash function, the energy consumption per day is 0.0045 J and 0.735 J by the design in [11] and our design, respectively. From our calculation, when integrating the accelerator in [11] and our proposed cryptographic accelerator, the endurance time of the ECG SoC is estimated to 53.33 days and 203.55 days, respectively. The results show that the ECG SoC integrated with our proposed cryptographic accelerator can significantly increase the endurance time by more than ten times.

For the application scenarios where both decryption and hash authentication are required to operate simultaneously, the latency of the proposed reconfigurable cryptographic accelerator is much lower than the state-of-the-art designs in [10–12]. The comparison of processing latency with [11] is performed because only the work in [11] presents the cycles of mode configuration and algorithm acceleration. This comparative study uses the following scenario as an example: the cryptographic SoC receives a 512-bit ciphertext and a 256-bit hash value simultaneously, and, therefore, it needs to use AES-128 and SHA-256

for decryption and hash authentication, respectively. For the design in [11], the reconfigurable cryptographic accelerator takes 367 cycles to be configured in the AES decryption mode, 80 (20×4) cycles to perform decryption, 370 cycles to be configured in the SHA-256 mode, and 160 (80×2) cycles to perform hash authentication. In total, it takes 977 cycles for the design in [11] to complete the decryption and hash authentication tasks in the aforementioned scenario in practical IoT applications. For our reconfigurable cryptographic accelerator design, it takes only two cycles to configure the reconfigurable cipher unit and reconfigurable hash unit to the AES-128 decryption mode and the SHA-256 mode through the AXI bus, respectively. Subsequently, the reconfigurable cipher unit takes 80 (20×4) cycles to perform decryption and the reconfigurable hash unit takes 160 (80×2) cycles to perform hash authentication simultaneously. As compared to the existing reconfigurable design for both block cipher and hash function algorithms in a single unified data path, our proposed design is able to complete the acceleration of computing block cipher and hash function algorithms at the same time by much shorter processing latency (i.e., a total of 162 cycles in the above case study), which can significantly reduce the number of processing cycles (i.e., 83% reduction as compared to the state-of-the-art design in [11]).

In a summary, the above comparison and evaluation against the state-of-the-art designs has shown that the proposed reconfigurable cryptographic accelerator design have achieved higher area efficiency, better energy efficiency and faster processing speed, which is more suitable for power-constrained and cost-sensitive IoT applications.

5. Conclusions

The paper presents the design and analysis of a reconfigurable cryptographic accelerator that consists of a reconfigurable cipher unit and a reconfigurable hash unit with a unified cipher data path and a unified compression data path, respectively. The solution of mapping block ciphers and hash functions to two separate reconfigurable units proposed in this work significantly reduces the number of cycles, by 83% compared with a unified reconfigurable engine solution in the literature, when both block ciphers and hash functions are required to operate at the same time in many IoT application scenarios. The two unified data paths are proposed by reusing the key registers and operators to reduce the hardware overhead. The reconfigurable S-Box module based on $GF(((2^2)^2)^2)$ is designed and reused to further reduce the hardware overhead by more than 30% compared with the conventional LUT solution. Thanks to the hardware architecture with two separate reconfigurable units and the reconfigurable design methods mentioned above, the energy efficiency and area efficiency of the proposed design in 65-nm CMOS technology has achieved 441 Gbps/W and 37.55 Gbps/mm², respectively. The high energy efficiency and area efficiency show that the proposed reconfigurable cryptographic accelerator is very attractive for many IoT devices with limited battery and form factor including intelligent small mobile intelligent robots, wearable medical devices and miniature unmanned aerial vehicles.

Because cryptographic chips are at risk of side channel attacks [22], we will further increase the capability of resisting side channel attacks in the future to implement a more secure cryptographic chip.

Author Contributions: Conceptualization, G.Y. and C.W.; methodology, C.W., Z.Z. and Z.L.; literature search and review, X.Z. (Xvpeng Zhang), B.L., Y.Z., X.H. and Z.S.; writing—original draft preparation B.L. and X.Z. (Xvpeng Zhang); writing—review and editing C.W., B.L., X.Z. (Xvpeng Zhang), G.Y., K.-S.C., Z.S. and X.Z. (Xuecheng Zou); supervision, C.W. and X.Z. (Xuecheng Zou). All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by National Key R&D Program of China (2019YFB1310001) and in part by the Fundamental Research Funds of the Central Universities under Grant 2019KFYXJJS049.

Institutional Review Board Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lin, J.; Yu, W.; Zhang, N.; Yang, X.; Zhang, H.; Zhao, W. A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications. *IEEE Internet Things J.* **2017**, *4*, 1125–1142. [\[CrossRef\]](#)
2. Yang, Y.; Wu, L.; Yin, G.; Li, L.; Zhao, H. A Survey on Security and Privacy Issues in Internet-of-Things. *IEEE Internet Things J.* **2017**, *4*, 1250–1258. [\[CrossRef\]](#)
3. Buenrostro, E.; Cyrus, D.; Le, T.; Emamian, V. Security of IoT Devices. *J. Cyber Secur. Technol.* **2018**, *2*, 1–13. [\[CrossRef\]](#)
4. Bhuiyan, M.N.; Rahman, M.M.; Billah, M.M.; Saha, D. Internet of Things (IoT): A Review of Its Enabling Technologies in Healthcare Applications, Standards Protocols, Security, and Market Opportunities. *IEEE Internet Things J.* **2021**, *8*, 10474–10498. [\[CrossRef\]](#)
5. Chaves, R.; Kuzmanov, G.; Sousa, L.; Vassiliadis, S. Cost-Efficient SHA Hardware Accelerators. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2008**, *16*, 999–1008. [\[CrossRef\]](#)
6. Ao, T.; He, Z.; Rao, J.; Dai, K.; Zou, X. A Compact Hardware Implementation of SM3 Hash Function. In Proceedings of the 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications, Beijing, China, 24–26 September 2014; pp. 846–850.
7. Nishikawa, N.; Iwai, K.; Kurokawa, T. High-performance Symmetric Block Ciphers on Multicore CPU and GPUs. *Int. J. Netw. Comput.* **2012**, *2*, 251–268. [\[CrossRef\]](#)
8. Mathew, S.; Satpathy, S.; Suresh, V.; Anders, M.; Kaul, H.; Agarwal, A.; Hsu, S.; Chen, G.; Krishnamurthy, R. 340 mV–1.1 V, 289 Gbps/W, 2090-gate NanoAES Hardware Accelerator with Area-optimized Encrypt/decrypt $GF(2^4)^2$ Polynomials in 22 nm Tri-gate CMOS. In Proceedings of the 2014 Symposium on VLSI Circuits Digest of Technical Papers, Honolulu, HI, USA, 10–13 June 2014; pp. 1048–1058.
9. Mathew, S.; Sheikh, F.; Agarwal, A.; Kounavis, M.; Hsu, S.; Kaul, H.; Anders, M.; Krishnamurthy, R. 53 Gbps Native $GF(2^4)^2$ Composite-Field AES-Encrypt/Decrypt Accelerator for Content-Protection in 45 nm High-Performance Microprocessors. *IEEE J. Solid-State Circuits* **2011**, *46*, 767–776. [\[CrossRef\]](#)
10. Liu, L.; Wang, B.; Deng, C.; Zhu, M.; Yin, S.; Wei, S. Anole: A Highly Efficient Dynamically Reconfigurable Crypto-Processor for Symmetric-Key Algorithms. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2018**, *37*, 3081–3094. [\[CrossRef\]](#)
11. Deng, C.; Wang, B.; Liu, L.; Zhu, M.; Wu, Y.; Li, H.; Yin, S.; Wei, S. A 60 Gb/s-Level Coarse-Grained Reconfigurable Cryptographic Processor with Less Than 1-W Power. *IEEE Trans. Circuits Syst. II Express Briefs* **2020**, *67*, 375–379. [\[CrossRef\]](#)
12. Du, Y.; Li, W.; Dai, Z.; Nan, L. PVHArray: An Energy-Efficient Reconfigurable Cryptographic Logic Array with Intelligent Mapping. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2020**, *28*, 1302–1315. [\[CrossRef\]](#)
13. Nyberg, K. Generalized Feistel Networks. In Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Kyongju, Republic of Korea, 3–7 November 1996; pp. 91–104.
14. Kam, J.B.; Davida, G.I. Structured Design of Substitution-Permutation Encryption Networks. *IEEE Trans. Comput.* **1979**, *28*, 747–753. [\[CrossRef\]](#)
15. Coron, J.S.; Dodis, Y. Merkle-Damgård revisited: How to construct a hash function. In Proceedings of the Annual International Cryptology Conference, Santa Barbara, CA, USA, 14–18 August 2005; pp. 430–448.
16. Liu, Y. The Design of Reconfigurable AES/SM4 IP Core against Side-Channel Attack. Master’s Thesis, Nanjing University of Aeronautics and Astronautics, Nanjing, China, 2017.
17. Hao, L.; Wu, L.; Zhang, X. Design and Implementation of SM4 Block Cipher Based on Composite Field. *Microelectron. Comput.* **2015**, *32*, 16–20.
18. Canright, D. A Very Compact S-Box for AES. In Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems, Edinburgh, UK, 29 August–1 September 2005; pp. 441–455.
19. Energizer CR1220 Lithium Coin Battery Product Datasheet. Available online: <https://data.energizer.com/pdfs/cr1220.pdf> (accessed on 18 November 2022).
20. Myers, J.; Savanth, A.; Gaddh, R.; Howard, D.; Prabhat, P.; Flynn, D. A Subthreshold ARM Cortex-M0+ Subsystem in 65 nm CMOS for WSN Applications with 14 Power Domains, 10T SRAM, and Integrated Voltage Regulator. *IEEE J. Solid-State Circuits* **2016**, *51*, 31–44.
21. Zhou, J.; Wang, C. An Ultra-Low Power Turning Angle Based Biomedical Signal Compression Engine with Adaptive Threshold Tuning. *Sensors* **2017**, *17*, 1809. [\[CrossRef\]](#)
22. Shan, W.; Fu, X.; Xu, Z. A Secure Reconfigurable Crypto IC with Countermeasures against SPA, DPA, and EMA. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2015**, *34*, 1201–1205. [\[CrossRef\]](#)