Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 8458314, 24 pages
https://doi.org/10.1155/2022/8458314

Research Article

Lightweight and High-Performance Data Protection for Edge Network Security

Xiaojie Chen,[1] **Bin Li**,[2] **and Qinglei Zhou**[2]

[1]*State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China*
[2]*School of Computer and Artificial Intelligence, Zhengzhou University, Zhengzhou 450001, China*

Correspondence should be addressed to Bin Li; cctvlibin@163.com

Received 18 June 2021; Revised 30 October 2021; Accepted 4 January 2022; Published 22 February 2022

Academic Editor: Ding Wang

To strengthen the security of edge network data and reduce network latency, in this paper, we combine an advanced reduced instruction set computing machine (ARM) and a field programmable gate array (FPGA) to propose a lightweight ARM-FPGA computing architecture for edge network data security protection and acceleration. Firstly, the access control lists are set through FPGA to authorize and filter illegal data, thereby reducing the transmission and processing of invalid data on the network. Secondly, based on the principles of dynamization, diversification, and randomization, the initial random key is generated using a pseudo-random number generator and a scrambling factor, and the hash of the data packet value updates the key to ensure "one frame, one key." Then, an ARM microprocessor is used to monitor the working status of the system in real-time. If an abnormality is found, different disturbance factors are activated using key management to change the system running status and restore the stability of the system. Finally, pipeline technology and critical path optimization on FPGA are used to parallelize the underlying encryption algorithm and data compression algorithm to achieve high-speed memory communication and meet the performance requirements of different networks. The experimental and analysis results show that the computing architecture designed in this paper has high network encryption performance and can effectively prevent data leakage. In addition, the effectiveness of the architecture in terms of network delay and network load is verified.

1. Introduction

The rapid popularization of 5th-generation mobile communication technology (5G) networks will lead to the connection of a large number of edge devices to the internet, such as mobile devices, autonomous vehicles, security monitoring devices, and smart homes. According to relevant statistics and forecasts, by 2025, the number of devices connected to the internet will reach 500 billion [1], which will lead to an explosion of data between edge nodes and cloud servers. Such large amounts of data bring large overhead for limited network resources, and the security risks and privacy protection issues of edge networks will also increase. The results of a survey conducted by the Internet Crime Complaint Center (IC3) show that network security issues have brought huge economic losses [2]. Cyberspace is currently facing a variety of attack methods, such as malware infection, injection attacks with camouflage, and distributed denial of service (DOS) attacks, all of which make it difficult to take safety precautions on existing networks. Network security has become an issue of widespread concern among countries and academic institutions worldwide [3] and a topic of extensive research.

Traditional networks are static, similar, and deterministic in their composition, and it is easy for attackers to study their operating rules, identify security flaws, and conduct detection and continuous intrusions. Among the many types of attacks, network sniffing [4] is a major threat to network security. In a network sniffing attack, various key information transmitted on the access network is monitored and can be stolen or tampered with. In addition, there are a large number of security vulnerabilities in current network equipment and components. Attackers can use vulnerabilities in routing equipment to directly eavesdrop on user data in the core network [5]. In response to this problem, the American academic community has proposed a "game-

changing" moving target defense (MTD) [6] technology. By constantly and randomly changing deployment mechanisms and strategies, the difficulty of conducting an attack is increased, and the flexibility of the system is improved. Examples include internet protocol (IP) address changes [7–10], dynamic ports [11], dynamic 3-layer encryption schemes [12], full protocol stack randomization [13], security function virtualization [14], route randomization [15], and data virtualization [16]. In China, Wu proposed mimic security defense (MSD) [17], which dynamically and pseudo-randomly selects different equivalent execution entities under active and passive conditions to establish a variable execution environment and reduce the security risk of the system.

Among various network security defense measures, the hardware structure and operating system security are the foundation, and cryptography is the key technology. Network encryption technology can ensure the safe transmission of data and prevent network sniffing attacks. However, in traditional network encryption, the same key is used to encrypt all data packets during the key life cycle. If an attacker intercepts a large number of data packets encrypted with the same key, a ciphertext-only attack may be successfully implemented [18]. To combat ciphertext-only attacks, for some scenarios with high-security requirements, such as confidential systems and military departments, it is necessary to implement a dynamically variable key to encrypt each data packet, i.e., to use "one packet, one key" [19].

In the era of large-scale growth of edge network data, problems caused by data transmission, such as network delays, transmission efficiency, and network bandwidth congestion, will bring major challenges. Data compression technology can provide effective coding to reduce repetitive data under the premise of ensuring data reliability, thereby reducing the amount of network data. To meet the data transmission rate and real-time requirements of high-energy physics experiments under limited bandwidth, Truong et al. [20] developed a real-time lossless compression method based on incremental coding technology and implemented it efficiently on FPGA, which greatly improved the amount and speed of data processed in the experiment.

Edge networks and edge devices have high energy efficiency and high security requirements. Effective network security technologies mainly use the randomness and dynamics of the system to resist attacks, and they use encryption technology to protect data privacy. In this work, existing defense technology is combined with the computing characteristics of the edge network to propose a lightweight high-performance data protection scheme. The main contributions of our scheme can be summarized as follows:

(1) A lightweight encryption system structure for edge networks is proposed, which makes full use of the uncertainty caused by the randomness, dynamics, and diversity of key transformations, making it difficult for attackers to establish a continuous and reliable attack chain.

(2) Compression and encryption processing of edge network data are performed on FPGA. An ARM microprocessor is used to control the initialization of the key, and the disturbance factor of the key is configured using state monitoring to form the heterogeneous lightweight data protection system structure of ARM-FPGA. FPGA is used to control the access of data frames, filter some attacks, and increase the attack threshold.

(3) The ShangMi (SM) public key encryption algorithm SM2, the SM message digest algorithm SM3, and the SM symmetric encryption algorithm SM4 are optimized on FPGA to achieve high-performance data encryption processing. At the same time, FPGA realizes a multichannel parallel data compression algorithm using a serial-parallel structure, which reduces the amount of original data transmission and increases the network load.

(4) Incorporating the idea of blockchain, the SM3 operation is performed on the sent data frame using a one-way and nontamperable modification of the Hash function to obtain the hash value, and a nonlinear operation is used to update the key to realize the "one frame, one encryption" method for network data and effectively resist network sniffing and ciphertext-only attacks.

2. Related Works

Data protection mechanisms have been studied extensively worldwide. In terms of physical structure defense, Wang et al. [21] elaborated on each type of attack by examining 11 typical vulnerable protocols and suggested corresponding countermeasures for wireless sensor networks (WSNs). Okhravi et al. proposed a real-time migration of key applications across heterogeneous platforms. The mimic defense web server proposed by Tong et al. [22] uses dynamic heterogeneous redundancy (DHR) to establish a software layer and a data layer. Multilayer mimic defense, such as in the operating system layer, can effectively resist a variety of intrusion detection methods and attacks. Pavithran et al. proposed a novel cryptosystem based on deoxyribonucleic acid (DNA) cryptography and finite automata theory [23]. The proposed scheme can protect the system against numerous security attacks, such as brute force attacks, known plaintext attacks, differential cryptanalysis attacks, cipher text-only attacks, man-in-the-middle attacks, and phishing attacks. Zhang et al. [24] proposed a lightweight encryption scheme for mobile ad hoc networks based on network coding to improve the security of network transmission.

With the development of edge computing and cloud computing, security technologies for edge networks and mobile networks have developed rapidly. Dar et al. [25] presented a context-aware encryption protocol suite that selects an optimal encryption algorithm according to device specifications and the level of data confidentiality. Qiu et al. [26] proposed a provably secure three-factor authentication and key agreement (AKA) protocol based on extended chaotic maps for mobile lightweight devices by adopting the techniques of "fuzzy verifiers" and "honeywords." Wang

et al. [27] exploited the Rivest–Shamir–Adleman (RSA) cryptosystem's computational imbalance at the encryption side and decryption side and made full use of the computation and storage capability of the cloud center to design a cloud-aided, efficient user authentication scheme with forward secrecy for Industry 4.0. Zhao et al. [28] designed a privacy-preserving data aggregation scheme for edge computing-supported vehicular ad hoc networks (VANETs) based on bilinear pairing and Paillier homomorphic encryption that not only preserves the privacy of uploaded data but also realizes batch operations.

FPGA enables energy-efficient computing and is widely used in edge network devices. Many security applications have also been implemented in FPGAs to allow security applications to run in real time [29], such as firewalls and data on high-speed network packet scanning. Soliman et al. [30] added a new dimension of security using frequency hopping to generate a pseudo-random pattern for switching between 5 lightweight cryptographic ciphers and an internal configuration access port controller, which decreased area utilization and power consumption by 58% and 80%, respectively. Pontarelli and others introduced a high-speed FPGA network intrusion detection system (NIDS) [31, 32]. NIDS checks the traffic flowing in the network to detect malicious content, such as spam and viruses.

Cryptographic algorithms are a core security technology and are of great significance in network security. Related algorithms proposed in China have received widespread attention and applications. The scalar multiplication calculation can be performed only once when co-signing. Zhang et al. [33] proposed an efficient and secure two-party distributed signature protocol based on the SM2 signature algorithm for key replay attacks. When a valid signature is generated, there is no need to rebuild the entire private key. Fan et al. [34] proposed a multiengine synchronous work method to carry out the SM4 algorithm in cipher block chaining (CBC) mode and achieve high-speed data encryption storage in solid-state hard disks. Verification on FPGA confirmed that this method can meet the requirements of high-speed communication interfaces. Yang et al. [35] implemented the high-performance encryption algorithms SM4-XTS and SM2 and large-integer modular exponentiation operations on FPGA to address the high concurrency and security issues of big data applications.

Data compression technology is an effective solution to reduce network delay and network load problems. Wang et al. [36] proposed a two-color image encryption algorithm based on two-dimensional compressed sensing and wavelets that can effectively reduce the amount of data, simplify the key, and improve the efficiency of data transmission and key distribution. Fouad et al. [37] proposed a hybrid data compression algorithm to process data and then added RSA-encrypted data to the compressed data using steganography, thereby improving data security and reducing the amount of data transmission and storage space required.

In summary, existing research on network security mechanisms has shown that the complexity of the system will increase with the complexity of the defense mechanism. Chinese encryption standards play a key role in existing protection mechanisms. In addition, network load and storage problems are becoming increasingly prominent. Therefore, this article proposes a lightweight, high-performance edge network data processing solution for edge networks that combine mimic defense, cryptographic algorithms, and data compression to meet data privacy protection and increased network load requirements.

3. Edge Network Data Security Protection Scheme

3.1. Edge Network Data Protection Design. Edge computing is an important part of new network tasks and application scenarios. It includes any computing resource or network resource between the data source and the cloud data center [38], and the transmission between the edge device and the cloud is processed and important. The value of network data and the protection of edge network data are of great significance to the interests of countries and individuals. Edge network data face issues, such as data security, privacy protection, and computing energy efficiency. Therefore, this paper designs a lightweight, high-performance data protection structure for edge networks. As shown in Figure 1, the ARM-FPGA heterogeneous data processing structure comprises the input port A and output port B of network communication, ARM, and FPGA computing components for data processing, and high-speed storage double data rate (DDR) memory.

FPGA is the core algorithm layer and the core of the entire framework. It includes the SM2 public key encryption algorithm, SM3 message digest encryption algorithm, SM4 symmetric encryption algorithm, and key generation algorithm, which can meet different encryption applications. It also includes the Lempel-Ziv (LZ) algorithm implementation (LZ4) as a lossless data compression algorithm to reduce the amount of communication data. A and B are network interfaces. In practical applications, Gigabit and 10-Gigabit networks can be used according to the requirements. The ARM processor is mainly responsible for the control of the upper layer, including key management, key agreement, and state management. The separation of control and calculation in different processing units can reduce the degree of coupling at the system layer. DDR memory is a cache library for network data, and it is also a data exchange center between ARM and FPGA.

In the data processing mechanism implemented by ARM-FPGA, the data are mainly communicated, calculated, and stored in ARM, FPGA, and DDR hardware, and the data processing flow is shown in Figure 2.

The specific process steps of ARM-FPGA system data protection are as follows:

(1) The data frame is received from network port A and transmitted to FPGA through the I/0 interface. Then, FPGA decapsulates the data frame, divides the data into abnormal data, network protocol data, and data to be encrypted using the access control list (ACL), and it discards them. Forwarding and data encryption are performed from network port B.

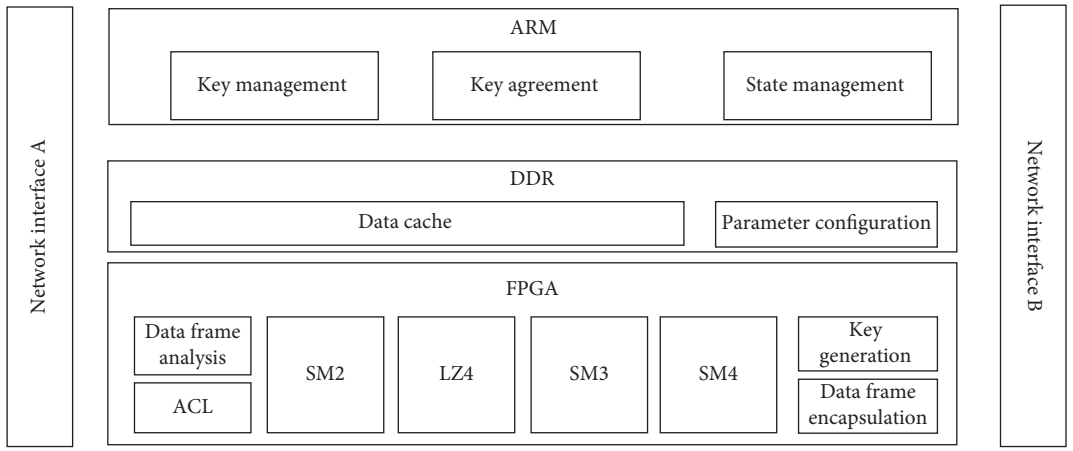

Figure 1: ARM-FPGA heterogeneous data processing structure.

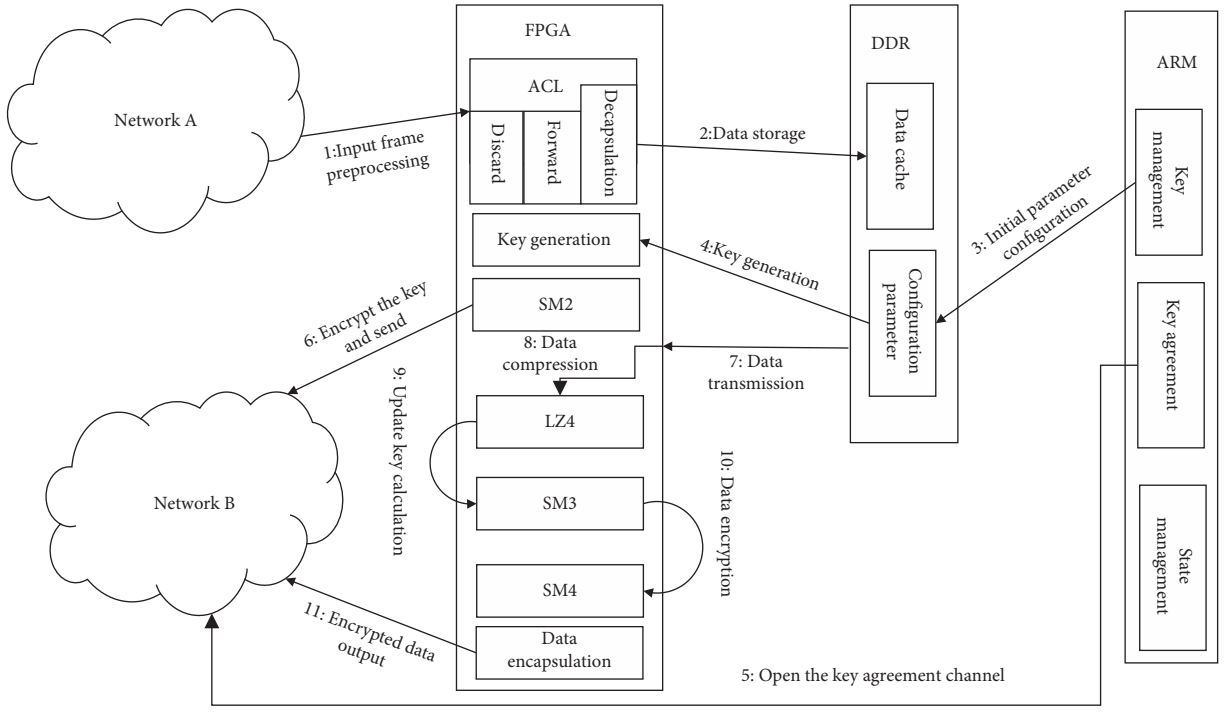

Figure 2: Data processing workflow.

(2) The encrypted data to be processed are written into DDR according to the preset value address, and routing information, such as the IP address, port number, and protocol type is cached.

(3) The ARM processor uses the key management control, starts the key generation module, and writes the initial configuration parameters into DDR.

(4) FPGA obtains the configuration parameters from DDR, configures the key generation algorithm, and generates and stores the key. The key updates the initial key according to a random timestamp to ensure the noncontinuity of the key.

(5) ARM opens a key agreement channel with the data-receiving peer using the upper-layer communication protocol.

(6) FPGA calls the high-speed SM2 module for key encryption and sends the initial encryption key to the opposite end for data decryption.

(7) FPGA and DDR establish a high-speed channel connection using the AXI bus protocol, and the data are transmitted to FPGA.

(8) FPGA calls the LZ4 algorithm to compress the data and buffers the processed data according to the first input first output (FIFO) FIFO_1 and FIFO_2.

(9) FPGA uses the SM3 algorithm to calculate FIFO_1 data using the empty and full signals, and the generated hash value and the initial key are calculated nonlinearly to obtain the key. The key of the current frame will also be used for the calculation in

the encryption key generation of the next frame to satisfy "one frame one key."

(10) The SM4 algorithm uses the data in FIFO_2 and the generated key to perform calculations to complete encryption.

(11) In FPGA, the data are encapsulated into a network frame according to the routing information and sent out using network port B.

During the entire data processing operation, ARM will discover abnormalities at runtime using state management and detection and reconfigure the key to protect the security of the system. Data compression is added in data processing, and the data need to be decompressed when they are transmitted to the opposite end. Therefore, a 1-bit compression flag and 1-byte frame identification (FID) are added to the data frame format. If the compression flag is 0, the compressed encrypted data have not been received. If the compression flag is 1, the last frame of data is compressed. The length of the 32-bit identification compression is added to the last frame of data. The frame sequence guarantees the correctness of the data sequence during decompression, and the modified encrypted frame structure is shown in Figure 3.

3.2. Key Generation and Configuration. The key is instrumental to ensuring data security. It is the most direct "key" for opening the encrypted data and the first part that an attacker will attack. Therefore, generating a secure key is the first step in data protection.

Pseudo-random number generators [39] have a wide range of applications in the fields of spread spectrum communication, information encryption, and system testing. The most commonly used method of generating pseudo-random numbers is to use a feedback shift register, which consists of two parts: the shift register and the feedback function. When the feedback function is a linear function, the feedback shift register is a linear feedback shift register (LFSR), as shown in Figure 4.

Here, fn is the feedback coefficient, 1 means the components are connected, and 0 means they are not connected. Clearly, the output sequence of LFSR is periodic, and an n-level LFSR provides at most $2n-1$ states (not including all 0 states). According to different feedback methods, the characteristic polynomial of LFSR can be defined.

$$p(x) = \sum_{i=0}^{n} f_i x^i = f_n + f_{n-1}x^{n-1} + \ldots + f_1 x + 1. \quad (1)$$

The scrambling factor can increase the scrambling effect and security of the algorithm. Cryptographic algorithms often use static parameters as the scrambling factor. To further increase the quality of the random number generated by LFSR, we perform the exclusive or (XOR) operation on the feedback value and the disturbance factor (DF), output the result, set the disturbance factors, including *FID*, *temperature*, and the random sequence (*RS*), and then, we perform the following operations:

$$\begin{aligned} randnum' &= FI\ D \oplus randnum(a_1, a_2, a_3, \ldots, a_n), \\ randnum' &= temperature \oplus randnum(a_1, a_2, a_3, \ldots, a_n), \\ randnum' &= RS \oplus randnum(a_1, a_2, a_3, \ldots, a_n), \end{aligned} \quad (2)$$

where $a_i (1 \le i \le n)$ represents LFSR status at the current moment. *FID* and *temperature* are real-time values that are used twice, which improves the utilization of data. *RS* is a random 8-row, 8-column, two-dimensional table generated by ARM, which is arranged in rows or columns and transmitted to FPGA using DDR memory. FPGA uses the lower 6 bits of *randnum* as the offset address to intercept the data and generate the scrambling factor. The *RS* example is shown in Figure 5.

This article defines 128-bit random numbers *randnum* and *randnum′*. There are 3 different LFSRs built into the random number generation scheme, which are randomly selected by FPGA, and the initial parameters are configured by ARM. The selection of the disturbance factor is dynamically configured by ARM according to the status information fed back by FPGA, which is divided into three following cases:

(i) When the encryption system is running normally, because of the stability of FPGA, the range of power consumption changes caused by the hardware operation is small. In this case, ARM configures the disturbance factor as *FID*. The system runs stably, and *FID* can change normally in real time.

(ii) When the encryption system is affected by external attacks or the environment, the temperature changes will be large. In this case, the configuration of the disturbance factor is *temperature*. It can stabilize the dynamics of the disturbance factor while simultaneously reducing the transmission of *FID*, changing the operating state of FPGA, and making the system operation stable again.

(iii) ARM detects that FPGA is running abnormally, and the temperature fluctuation range is small. At this time, the disturbance factor is configured to *RS*, and FPGA needs to receive only the disturbance factor, which reduces the number of FPGA calculations. *RS* is dynamically and randomly generated by ARM, which can ensure the dynamic randomness of the key.

Through the dynamic random combination of 3 kinds of pseudo-random number generators and 3 kinds of scrambling factors, the randomness, dynamics, and security of the key can be guaranteed, and the functions of the three scrambling factors are complementary to each other, which is beneficial for the stability of the system.

3.3. High-Speed Memory Data Transmission and Management. Because of the high computational complexity of data compression, the processing performance is low. Therefore, the data to be processed needs to be temporarily stored in memory to prevent network data congestion and loss because of mismatched data transmission

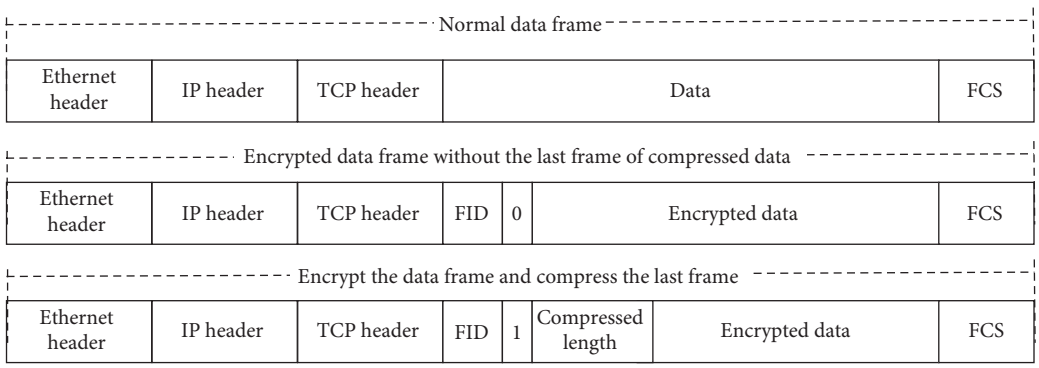

Figure 3: Data encryption frame format.

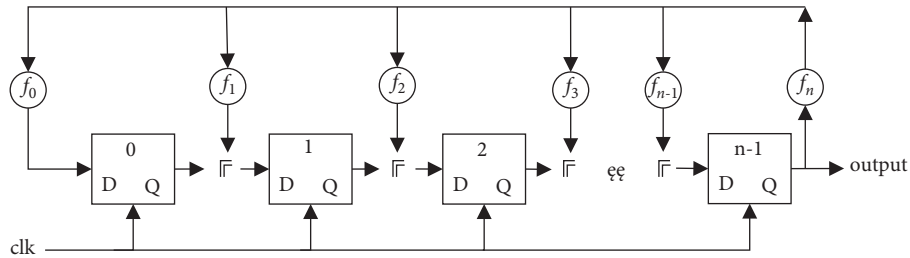

Figure 4: LFSR structure.

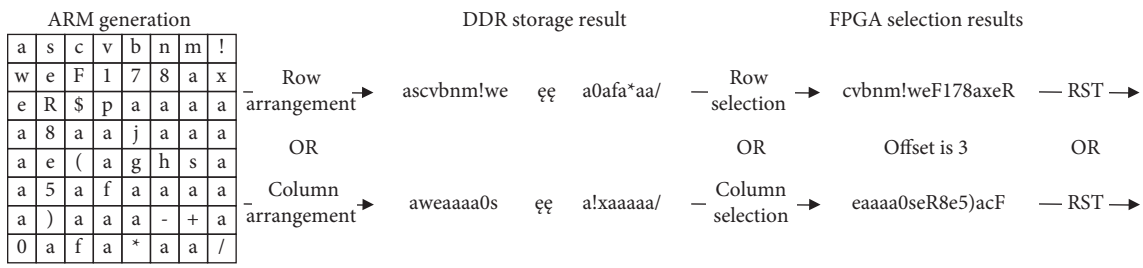

Figure 5: *RS* generation example.

and data processing in high-speed networks. In addition, data transmission between DDR and FPGA needs to meet high-performance requirements. Therefore, to achieve high-speed data exchange between FPGA and DDR, AXI4 (advanced extensible interface) protocol is adopted. The AXI4 protocol is the most important part of the advanced microcontroller bus architecture (AMBA) proposed by ARM and is mainly used in high-performance address mapping communication scenarios. The AXI4 communication protocol supports burst transmission, i.e., in data transmission, data can be obtained continuously. At a burst length of 16 and a data width of 256 bits, the amount of data transferred at one time is 512 bytes, i.e., one set of offset addresses corresponds to 16 sets of data, and FPGA and DDR are connected using memory interface generator (MIG) communication. Data transmission is carried out at the physical level. The memory communication design structure of the data to be compressed is shown in Figure 6.

The solid line in Figure 6 is the true transmission flow direction of data, and the dotted line points to the mapping position of the data in DDR. Data storage control and reading control are connected to MIG's user-side memory interface using the AXI4 protocol. The control signals include the read address, read data, read feedback, write address, write data, write response, and write feedback. The memory controller is based on read and write requests, and it stores or reads data from the physical layer interface in order. In the memory communication design structure, the data transmission between the modules is carried out using a FIFO buffer so that the memory communication and data calculation are separated. The storage control module, the read control module, and the compression module are independent of each other and are calculated in parallel, thereby reducing the degree of data coupling and delay in data communication.

ARM is the FPGA controller. The key in FPGA needs to be configured, and FPGA can be customized in real time using the running status fed back by FPGA. The communication data between ARM and FPGA needs to be cached in DDR and controlled by the direct-memory-access (DMA) controller. The structure is shown in Figure 7.

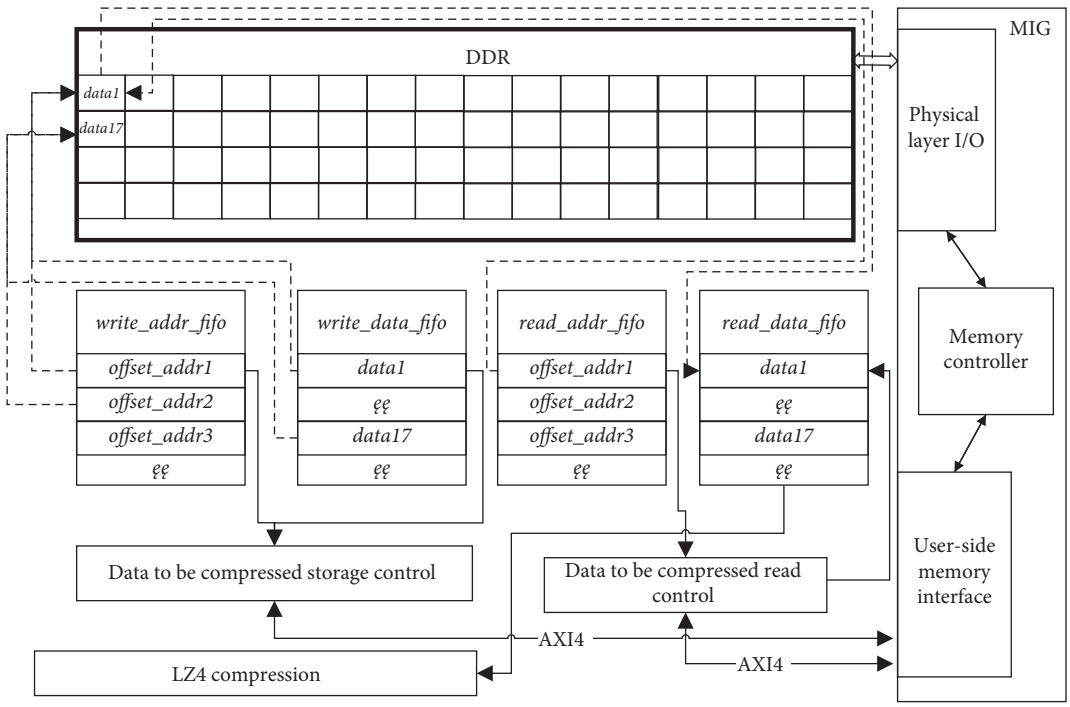

Figure 6: High-speed memory data communication structure.

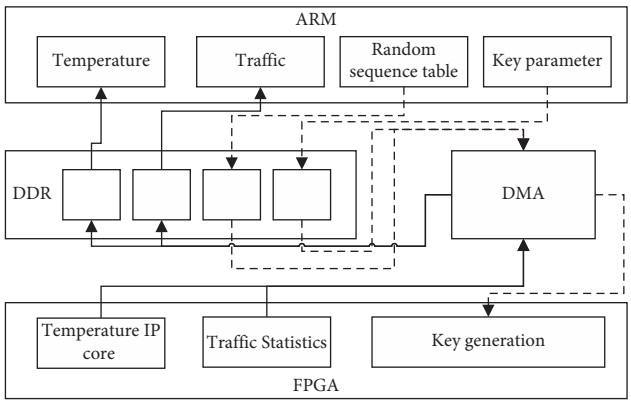

Figure 7: ARM and FPGA data communication structure.

The solid line in Figure 7 shows that FPGA transmits temperature and flow data to ARM using the temperature IP core and flow statistics module. The dotted line is the key configuration data of ARM to FPGA, and different addresses are allocated for different data in DDR. The read and write operations of different data in FPGA are calculated in parallel, and the arbitration transmission is carried out using DMA to ensure that the data can be transmitted to the correct address. ARM needs to start a DMA transmission channel to read and write data from DDR, and it operates on different data by polling tasks.

3.4. Fine-Grained Data Access Control. Access control allows legitimate users to access the operating data using authorization, and unauthorized users are prohibited from accessing the operating data illegally. An ACL is an important access authorization strategy that filters illegal users using preset list attributes, thereby playing a role in data protection. The data transmission of edge network equipment uses clear and trusted network information, such as an IP address, port number, and transmission protocol. Therefore, this article presents an access control list, including data frame filtering attributes, such as the source media access control (MAC) address, destination MAC address, source IP address, destination IP address, source port number, and destination port number. When a data frame is transmitted to FPGA, the data frame is parsed and filtered according to frame information and preconfigured ACL. The implementation structure is shown in Figure 8.

The network data frame is transmitted to FPGA. Firstly, the data frame is decapsulated, and the frame information and frame data are buffered in the FIFO buffer. Then, the ACL module reads the data frame information and static access control attribute information to filter, authorize, and divide the data. There are three kinds of processing: (1) discard: discard the frame information, and read the corresponding data from the data FIFO buffer to discard them, (2) forwarding: buffer the frame information, read the data from the data FIFO buffer, and send them to the data frame encapsulation module to process and output them using the network sending port, and (3) encryption: buffer the frame information, read the data in the data FIFO buffer, cache them in the DDR using the high-speed interface, process the data according to the encryption mechanism, send the processed ciphertext and re-encapsulate, and send the frame information.

Using ACL for access control can directly filter illegal data and reduce data transmission. Legal data are processed in two ways, namely forwarding and encryption, which can

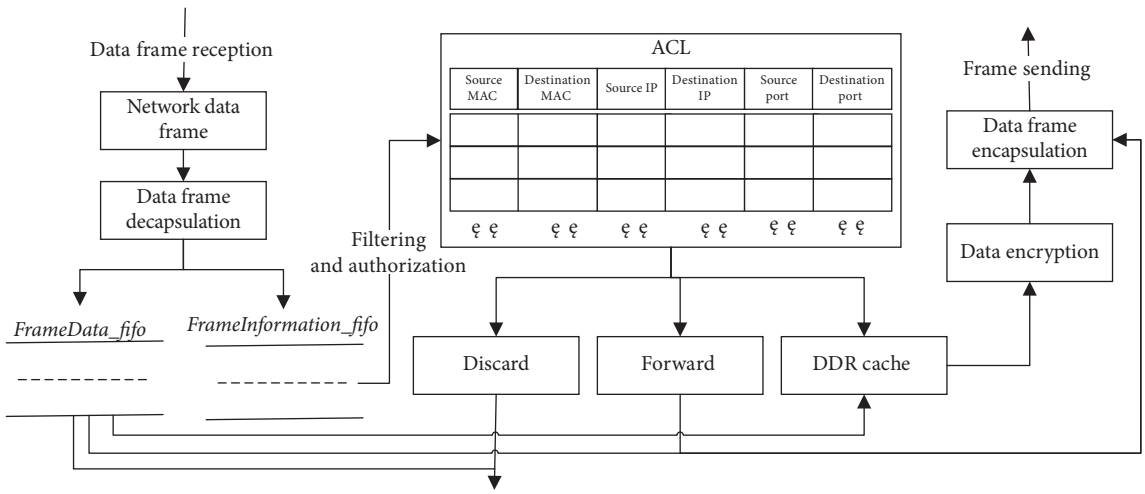

Figure 8: Data access control structure.

reduce the amount of calculation needed for data processing, thereby reducing the energy consumption of the device and simultaneously ensuring the privacy and security of important data.

4. High-Performance Core Algorithm Implementation

Encryption algorithms and compression algorithms consume the most computing resources and the longest computing time in the entire framework. FPGA is the main computing component. High-performance cryptographic algorithms and data compression algorithms are essential for improving edge network data protection and reducing network latency. Therefore, one of the key research objects of this article is the parallelization of cryptographic algorithms and data compression algorithms on FPGAs based on pipeline technology, storage optimization, prime domain computing optimization, collaborative computing, multichannel parallelism, and other methods to obtain high-performance, low-energy algorithms.

4.1. SM4 Parallel Implementation. The SM4 cipher algorithm is a block encryption algorithm. The input data and key are 128 bits, and the encryption operation adopts 32 rounds of nonlinear iterative structure, which can be guaranteed in terms of security, as there is still no effective method of attacking it in a limited time [40]. The SM4 encryption algorithm includes two parts: key expansion and round function encryption. The specific algorithm flow is shown in Algorithm 1.

Step 1 of the encryption algorithm sets the initialization parameter, and the initial key is obtained using the system parameter. Steps 2 to 10 are 32 rounds of loop iterations, in which steps 3 to 5 and 6 to 9 generate each round of the subkey rk_i and data encryption using exclusive OR operations, S-box permutations, and linear transformations. These steps encrypt and scramble the data, thereby improving the resistance to attack. Finally, Step 11 obtains the final output using the reverse-sequence operation. The operation and symbol descriptions for the algorithm are shown in Table 1.

The SM4 encryption algorithm has a small amount of calculation and low complexity in the initialization key and reverse-sequence output generation stages and requires only one calculation clock. Both subkey generation and data encryption require round function calculations, including XOR operations with four operands, four S-box lookups, linear transformations, and XOR operations, where the S-box lookups need 256 data points, i.e., obtaining an 8-bit input requires 256 corresponding 8-bit data points, and its complexity will affect the timing implemented on FPGA, which will make it difficult to improve the performance of the algorithm. Therefore, it is necessary to optimize the round function and S-box.

The round function is the key path of the SM4 algorithm. The main computing resources in FPGAs include look-up tables (LUTs), flip-flop (FF) registers, and random access memory (RAM). If the critical path is implemented in one clock, a combinational logic method is required, which will consume a large number of LUTs and calculations at the same time. The frequency will also drop, and the relationship between LUT and FF resources is 1 : 2. When implementing FPGAs, it is necessary to ensure the balanced utilization of resources and maximize the calculation frequency. We separate the critical path and use sequential logic to implement A and X_{i+4}, which consumes 2 clock cycles. Four S-box operations can be processed in one clock cycle in parallel. Because of the need to implement a variety of algorithms in this solution and the fact that the RAM resource occupancy is small, the S-box data are stored in RAM to reduce the use of LUTs. When operating, only the searched position needs to be input as an address, which can be completed in one clock. The round function calculation structure of the optimized conversion is shown in Figure 9.

The key to maximizing the performance of the SM4 algorithm is to improve the algorithm throughput and realize the SM4 algorithm for the full pipeline. The SM4 algorithm contains 32 rounds of iterative operations. The critical path of each round of operation is split into 3 clock

SM4 Encryption Algorithm
Input: $X = (X_0, X_1, X_2, X_3)$// 128-bit data
$\mathrm{MK} = (\mathrm{MK}_0, \mathrm{MK}_1, \mathrm{MK}_2, \mathrm{MK}_3)$// 128-bit key
Output: Y
(1) $(K_0, K_1, K_2, K_3) = (\mathrm{MK}_0 \oplus \mathrm{FK}_0, \mathrm{MK}_1 \oplus \mathrm{FK}_1, \mathrm{MK}_2 \oplus \mathrm{FK}_2, \mathrm{MK}_3 \oplus \mathrm{FK}_3)$
(2) **for** $i = 0$ **to** 31 **do**
(3) $B = (b_0, b_1, b_2, b_3) = K_{i+1} \oplus K_{i+2} \oplus K_{i+3} \oplus \mathrm{CK}_i$
(4) $T' = (\mathrm{Sbox}\ (b_0), \mathrm{Sbox}\ (b_1), \mathrm{Sbox}\ (b_2), \mathrm{Sbox}\ (b_3))$
(5) $rk_i = K_{i+4} = K_i \oplus T' \oplus (T' <<< 13) \oplus (T' <<< 23)$
(6) $A = (a_0, a_1, a_2, a_3) = X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i$
(7) $\tau = (\mathrm{Sbox}\ (a_0), \mathrm{Sbox}\ (a_1), \mathrm{Sbox}\ (a_2), \mathrm{Sbox}\ (a_3))$
(8) $T = \tau \oplus (\tau <<< 2) \oplus (\tau <<< 10) \oplus (\tau <<< 18) \oplus (\tau <<< 24)$
(9) $X_{i+4} = X_i \oplus T$
(10) end for
(11) $Y = (Y_0, Y_1, Y_2, Y_3) = (X_{35}, X_{34}, X_{33}, X_{32})$

Algorithm 1: SM4 encryption algorithm.

Table 1: SM4 algorithm parameter description.

Operator	Description
$\oplus$	32-bit XOR
$\lll i$	32-bit circular shift left i-bit
FK_i	System parameters
CK_i	Fixed parameters
Sbox ()	S-box replacement, 8-bit input, 8-bit output

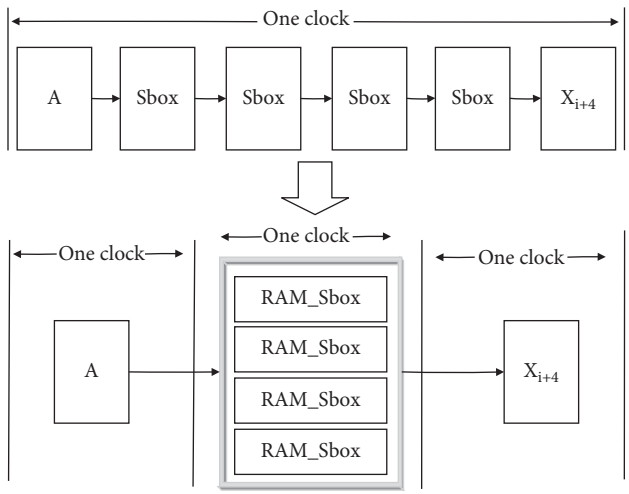

Figure 9: Round function optimization structure.

cycles, i.e., key generation and data encryption require 3 clocks, and a total of 6 clock cycles are required for one round of calculation. The transfer of data follows a similar process. Therefore, in a single round of key generation and data encryption, a three-stage pipeline is used to obtain 98-stage pipelines, including the two-stage operations of preprocessing and reverse-sequence output. The structure of SM4 algorithm is shown in Figure 10.

When calculating the data to be encrypted, the continuous data to be encrypted are first input into *Round0*, the first sub-key is calculated after three clock cycles, and the value of A_0 is calculated from the output. At the same time, the second group of data outputs the first sub-key using parallel calculations. After two more clock cycles, key expansion and round encryption enter the parallel calculation state. After the 99th clock cycle, the first group of data is encrypted and output. The computing module occupied by the 96-stage pipeline runs at full load. The edge network needs to encrypt a large amount of data. The continuous data are input into the encrypted data stream. After 99 clock cycles, each clock has an output that can ensure the continuity of the ciphertext data and meet the network delay requirements, and throughput reaches its maximum.

4.2. SM3 Parallel Implementation. The hash algorithm has a wide range of applications in cryptography and data encryption and decryption [41–43]. SM3 is a hash algorithm proposed in China. It is suitable for digital signatures and verification in commercial cryptographic applications, message authentication code generation, verification, and randomization. The generation of numbers has the ability to resist currently known attacks more than the secure hash algorithm 256 (SHA-256) [44]. The SM3 algorithm inputs arbitrary-length data. After filling and iterative compression, it produces fixed-length 256-bit data.

Data padding adds input data m to data m' with a length of 512 bits according to the rule and divides the data into blocks of 16 words with a length of 32 bits: $W_0 \sim W_{15}$. The core of the algorithm is 64 rounds of iterative compression operations, and the compression function operation in each round is shown in Figure 11(a). The initial values of A through H in Figure 11 are the system vector V, and t represents the number of calculation rounds. After 64 rounds of calculation, the results $A_{63} \sim H_{63}$ and the initial vector V are XORed to obtain the final result. The SM3 algorithm parameter description is shown in Table 2.

FPGA is mainly composed of logic gates. Operations, such as AND, OR, NOT, and shifting have greater advantages, however, the delay of addition operations is relatively high. The longest path of the compression function in the SM3 algorithm is to calculate the values of A and E, which requires 5 addition operations. In the algorithm, SS1 is shared by A and E, and SS2 requires only one step of an

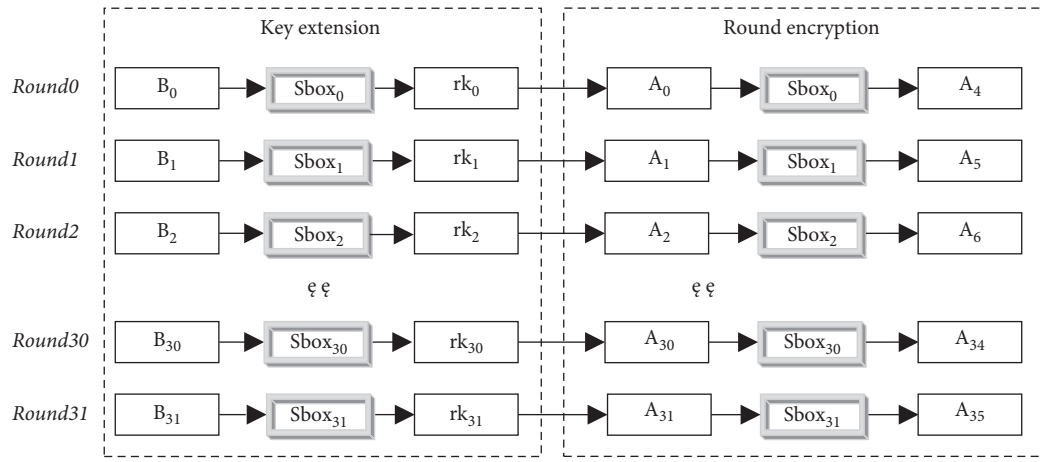

Figure 10: Structure of SM4 algorithm rounds.

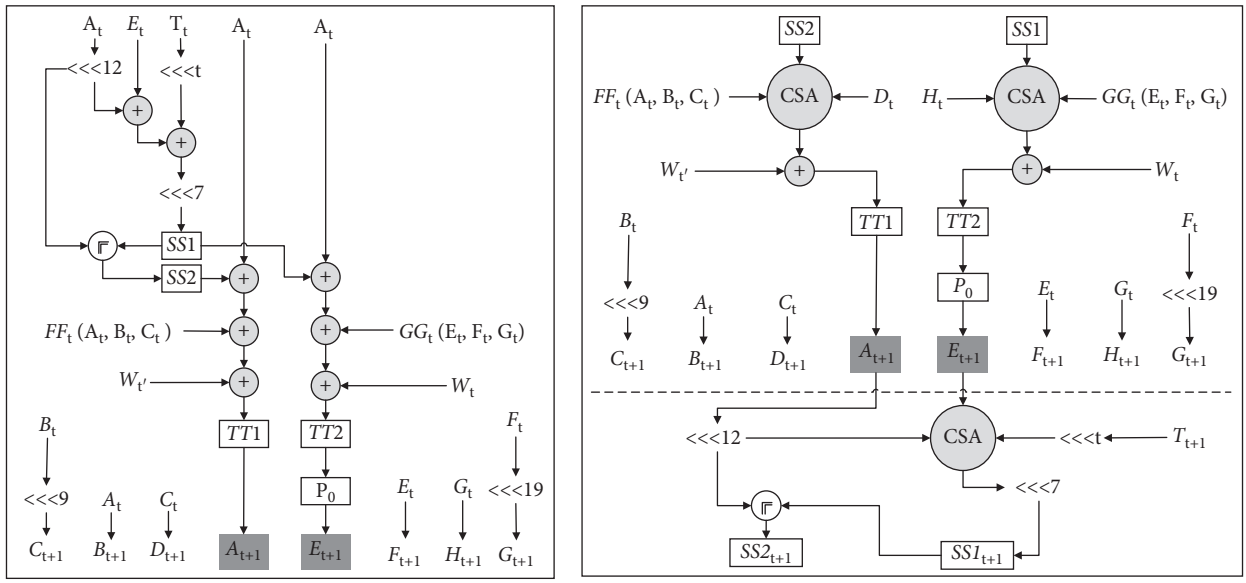

Figure 11: SM3 compression function structure. (a) Initial structure. (b) Optimized structure.

exclusive OR operation on SS1 and the value of the operation. There is no coupling with subsequent operations. Therefore, the critical path needs to be truncated. SS1 and SS2 are precalculated using a clock, and the critical path is reduced to 3 addition operations, which can reduce the algorithm delay.

Carry-save address (CSAs) can convert the addition of three operands into one addition operation and 8-bit operations using logical bit operations. The CSA operation is defined as CSA $(x, y, z) = (x \hat{} y \hat{} z) + (((x \& y) \mid (x \& z) \mid (y \& z)) << 1)$. There are multiple addition operations in the SM3 algorithm. Hence, CSAs can be used for conversion. The conversion calculation process is as follows:

$$\begin{aligned}(A_t \lll 12) + E_t + (T_t \lll t) &= \mathrm{CSA}((A_t \lll 12), E_t, (T_t \lll t)),\\ FF_t(A_t, B_t, C_t) + D_t + \mathrm{SS2} &= \mathrm{CSA}(FF_t(A_t, B_t, C_t), D_t, \mathrm{SS2}),\\ GG_t(E_t, F_t, G_t) + H_t + \mathrm{SS1} &= \mathrm{CSA}(GG_t(E_t, F_t, G_t), H_t, \mathrm{SS1}).\end{aligned} \quad (3)$$

By precalculation and CSA conversion, the key path of the SM3 algorithm changes from 5 addition delays to 2 addition delays, and the calculation delay per unit time is greatly reduced.

Through precalculation, the critical path is split into two parts, and SS1 and SS2 need the values of A and E in the current round to participate in the calculation. If the single-round internal compression function is divided into a two-stage pipeline calculation, at least a 128-stage pipeline is required. A pipeline with more than 100 stages will have a greater timing impact on FPGA chips with a lower process level. Therefore, the calculation of SS1 and SS2 in round $t + 1$ is performed after the compression function of round t, and they are in the same clock. It can reduce the impact of timing without increasing the delay of the critical path. After precalculation and CSA optimization, the single-round processing structure is shown in Figure 11(b). The value of

Table 2: SM3 algorithm parameter description.

Operator	Description	Calculation
T_t	Constant	0x79cc4519 $0 \le t \le 15$ 0x7a879d8a $16 \le t \le 63$
FF_t (X, Y, Z)	Boolean function	$X \oplus Y \oplus Z$ $0 \le t \le 15$ $(X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$ $16 \le t \le 63$
GG_t (X, Y, Z)	Boolean function	$X \oplus Y \oplus Z$ $0 \le t \le 15$ $(X \wedge Y) \vee (\neg X \wedge Z)$ $16 \le t \le 63$
P_0 (X)	Permutation function	$X \oplus (X \lll 9) \oplus (X \lll 17)$
P_1 (X)	Permutation function	$X \oplus (X \lll 15) \oplus (X \lll 23)$
W_t	Extended word	P_1 $(W_{t-16} \oplus W_{t-9} \oplus (W_{t-3} \lll 15)) \oplus (W_{t-13} \lll 7) \oplus W_{t-6}$ $16 \le t \le 67$
W_t'	Extended word	$W_t \oplus W_{t+4}$

W involved in the calculation includes the values generated by the different iteration rounds, and multiple W values need to be stored. There is only one-way data transmission with the compression function, which can be used as a separate first-stage pipeline. The SM3 compression function requires 64 iterations. Therefore, a 65-stage full pipeline is used for implementation.

4.3. High-Speed SM2 Algorithm Implementation. The SM2 algorithm is a public key encryption algorithm based on the elliptic curve discrete logarithm problem. In the prime domain, an elliptic curve E (Fp) is described as [45] $y^2 = x^3 + ax + b \pmod{p}$. The security of an elliptic curve cryptosystem is mainly based on the difficulty of finding the inverse of the point multiplication. Point multiplication, also called a multiple-point operation, refers to the multiplication operation of a base point P on a curve with an integer k, i.e.,

$$kp(x) = \sum_{1}^{k} p = p + p + p + \cdots + p. \tag{4}$$

The point multiplication process includes point doubling and point addition operations. Hence, the optimization of point addition, point doubling, and point multiplication is an important means of improving the efficiency of elliptic curve calculation.

The SM2 algorithm uses the reconfigurable features of FPGA, combined with Karatsuba-Ofman algorithm (KOA) multiplication, fast modular reduction, radix-4 modular inversion, Montgomery point multiplication, point addition, point doubling, and other optimization methods, to achieve high energy efficiency and resistance to attacks. By the bottom-up design method, the most basic operations at the bottom are realized with modular addition and subtraction, modular reduction, modular inversion, and modular multiplication. Then, the multiplication operation is optimized by point doubling and point addition, and finally, the SM2 encryption function is realized. The overall structure is shown in Figure 12.

When calculating the point multiplication, the point addition and point doubling operations are called multiple times, and the coordinate conversion is calculated only once at the end. Therefore, the point addition and point doubling operations are optimized for the best performance, and coordinate conversion is optimized for resource use. Secondly, the master control state machine performs scheduling and management of the dot product module to meet the calculation requirements of different functions. Finally, coordinate conversion shares the modular addition and subtraction module to reuse resources to reduce the consumption of FPGA resources.

4.3.1. KOA Fast Multiplication. The core idea of the KOA [46] algorithm is to "divide and conquer." It uses a recursive method to decompose a complex multiplication operation into multiple simple multiplication operations, which is faster and more efficient than traditional calculations. If two numbers of length n are directly multiplied, the complexity is $O(n^2)$. Using the KOA algorithm can reduce the complexity to $O(n^{\log_2 3})$.

For SM2, the parameter is 256 bits, and the FPGA digital signal processor (DSP) supports multiplication operations with a maximum width of 64 bits. Then, 256 bits can be divided into 128 bits, and 128 bits can be divided into 64 bits. After two recursive operations, the final result is obtained, as shown in Algorithm 2.

To further optimize the implementation of the KOA algorithm on FPGA, the 64-bit DSP multiplication clock cycle is set to 0, and the various modules are interconnected using wire type variables. When 256-bit data are input, the result can be calculated immediately and output by the register buffer. The whole calculation process takes 1 clock cycle.

4.3.2. Fast Modular Reduction. For fast modular reduction, several addition and subtraction operations can be used to obtain the modular reduction result. Compared with the currently universal Montgomery algorithm, fast modular reduction saves several 256-bit multiplication operations, and the performance can be significantly improved. If

$$p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1, \tag{5}$$

then for a large number A, we have the following:

$$A = A_{15} \times 2^{480} + A_{14} \times 2^{448} + \cdots + A_1 \times 2^{32} + A_0. \tag{6}$$

Each A_i is a 32-bit integer. Then, A can be expressed as follows:

$$A = A_{15} \parallel A_{14} \parallel + \cdots + A_1 \parallel A_0. \tag{7}$$

Then,

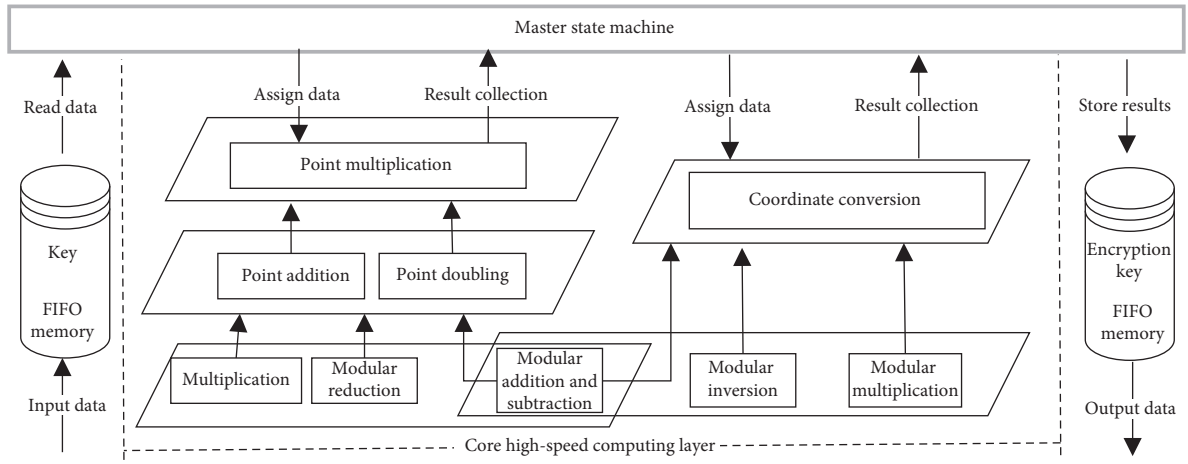

Figure 12: Overall architecture of the SM2 algorithm.

$$B = A \bmod p = (2S_0 + 2S_1 + 2S_2 + 2S_3 + 2S_4 + S_5 + S_6 + S_7 + S_8 + S_9 - D_1 - D_2 - D_3 - D_4) \bmod p, \tag{8}$$

where each 256-bit operand is represented as shown in Table 3.

4.3.3. Extended Euclidean Modular Inverse. The extended Euclidean algorithm uses the toss and turns division method to obtain the modular inverse. According to the nature of the common divisor, all divisions can be changed to addition and subtraction operations, and the division-by-2 operation is completed by binary shift, which is conducive to hardware implementation. Here, the algorithm is expressed in a quaternary system using state machine cycle control to optimize the extended Euclidean algorithm, and the value of $b/a \bmod p$ can be obtained directly. The specific process is shown in Algorithm 3.

In Algorithm 3, the first line is the initialization state. The second line is the loop judgment state. Lines 3–27 are for various judgments and calculations. The last line is the output. Secondly, the calculation of u and v always ensures that the result value is less than p, and no additional processing is required. The addition and subtraction of x_1 and x_2 may be greater than p or may overflow, and division by 2 and division by 4 require additional judgment and calls of modulo addition and subtraction for processing. The specific calculation formula is as follows:

$$\frac{x}{2} \bmod p = \begin{cases} x \gg 1, & \text{if } x[0] == 1'b0, \\ x \gg 1 + p \gg 1 + 1, & \text{if } x[0] == 1'b1, \end{cases}$$

$$\frac{x}{2} \bmod p = \begin{cases} x \gg 2, & \text{if } x[1:0] == 2'b00, \\ (x \gg 1 + p \gg 1 + 1) \gg 1, & \text{if } x[1:0] == 2'b01, \\ x \gg 2 + p \gg 1 + 1, & \text{if } x[1:0] == 2'b10, \\ (x \gg 1 + p \gg 1 + 1) \gg 1 + p \gg 1 + 1, & \text{if } x[1:0] == 2'b11. \end{cases} \tag{9}$$

4.3.4. Point Multiplication Optimization. At present, the Montgomery point multiplication algorithm is the most efficient and widely used algorithm [47, 48]. The specific operation process is shown in Algorithm 4.

As shown in Algorithm 4, regardless of the value of k_i, the point addition and point doubling operations will be calculated during each cycle, and the two are independent of each other and can be executed in parallel. At the same time, because of the simultaneous calculation of point addition and point doubling, the power consumption information leaked during the point multiplication operation is unruly, which can effectively resist a simple power consumption attack (SPA).

To improve the calculation efficiency of point addition and point doubling, Montgomery point multiplication needs only the x coordinate to participate in the calculation and calculate the y coordinate in the last step, which greatly simplifies the calculation process.

KOA multiplication
Input: A, B, n
Output: C
(1) **if** ($n == 64$) return $C = A \times B$;
(2) $A = A^H \times 2^{n/2} + A^L$
(3) $B = B^H \times 2^{n/2} + B^L$
(4) $C_1 = \text{KOA}\ (A^H, B^H, n/2)$
(5) $C_2 = \text{KOA}\ (A^L, B^L, n/2)$
(6) $C_3 = \text{KOA}\ (A^H + A^L, B^H + B^L, n/2)$
(7) $C = C_1 \ll n + (C_3 - C_2 - C_1) \ll (n/2) + C_2$

Algorithm 2: KOA multiplication.

Table 3: Fast modulus reduction of various parameters.

	255–224	223–192	191–160	159–128	127–96	95–64	63–32	31–0
S_0	A_{15}	*0*	A_{15}	A_{14}	A_{13}	*0*	A_{15}	A_{14}
S_1	A_{14}	0	0	0	0	0	A_{14}	A_{13}
S_2	A_{13}	0	0	0	0	0	0	A_{15}
S_3	A_{12}	0	0	0	0	0	0	0
S_4	A_{15}	A_{15}	A_{14}	A_{13}	A_{12}	0	A_{11}	A_{10}
S_5	A_{11}	A_{14}	A_{13}	A_{12}	A_{11}	0	A_{10}	A_9
S_6	A_{10}	A_{11}	A_{10}	A_9	A_8	0	A_{13}	A_{12}
S_7	A_9	0	0	A_{15}	A_{14}	0	A_9	A_8
S_8	A_8	0	0	0	A_{15}	0	A_{12}	A_{11}
S_9	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
D_1	0	0	0	0	0	A_8	0	0
D_2	0	0	0	0	0	A_9	0	0
D_3	0	0	0	0	0	A_{15}	0	0
D_4	0	0	0	0	0	A_{14}	0	0

In the standard projective coordinate system, for points $P(X_1, Y_1, Z_1)$ and $Q(X_2, Y_2, Z_2)$, the calculation formulas for point addition and point doubling are as follows:

$$\begin{cases} X(P+Q) = (X_1X_2 - aZ_1Z_2)^2 - 4bZ_1Z_2(X_1Z_2 + X_2Z_1), \\ Z(P+Q) = x_G(X_1Z_2 - X_2Z_1)^2, \\ X(2P) = \left(X_1^2 - aZ_1^2\right)^2 - 8bX_1Z_1^3, \\ Z(2P) = 4Z_1\left(X_1^3 + aX_1Z_1^2 + bZ_1^3\right)(X_1Z_2 - X_2Z_1)^2. \end{cases} \tag{10}$$

Upon converting the result to affine coordinates, we have the following:

$$\begin{cases} x_1 = \dfrac{X_1}{Z_1}, \\ x_2 = \dfrac{X_2}{Z_2}, \\ y_1 = \dfrac{2b + (a + x_Gx_1)(x_G + x_1) - x_2(x_G - x_1)^2}{2y_G}. \end{cases} \tag{11}$$

Then, the point (x_1, y_1) is the result. Here, (x_G, y_G) are the coordinates of the base point G.

Under standard projection coordinates, the projection point and the affine point will be mapped one by one. At the beginning of the operation, the affine coordinates will be transformed to projection coordinates, and they will be mapped back to affine coordinates at the end of the operation. Therefore, in the entire calculation process, only one modular inverse operation is used at the end, and there is no modular inversion in the intermediate iteration process.

Finally, to further optimize the calculation efficiency of point addition and point doubling, the data flow is deeply optimized so that the calculation can be completed in the shortest time. Since fast modular multiplication consists of two modules, namely KOA multiplication and fast modular reduction, and both can calculate the result within one clock cycle, part of the calculation process of adjusting point addition and point doubling alternately calls KOA multiplication and fast modular reduction modules, taking full advantage of computational efficiency. After optimization, the calculation of point addition and point doubling can be completed in 12 clock cycles, indicating very high efficiency.

4.4. Parallel LZ4 Algorithm Implementation. One of the fastest compression algorithms is the LZ4 algorithm proposed by Yann Collet in 2012 [49], which is appropriate for data compression applications of lightweight hardware devices. The LZ4 algorithm inputs 1 byte of data *char* each time. After data compression and LZ4 encoding processing, the compression stage is subdivided into four parts: dictionary matching, optimal matching filtering, discarding of matching data, and data separation. The flow of the LZ4 algorithm is shown in Figure 13.

Extended Euclidean modular inverse
Input: a, b, p
Output: $c = b/a \bmod p$
(1) $u = a$; $v = p$; $x_1 = b$; $x_2 = 0$
(2) **while** $(v > 0)$
(3) **if** $(u[1:0] == 2'b00)$
(4) $u = u \gg 2$, $x_1 = x_1/4 \bmod p$
(5) **else if** $(V\ [1:0] == 2'b00)$
(6) $v = v \gg 2$, $x_2 = x_2/4 \bmod p$
(7) **else if** $(u\ [1:0] == v[1:0])$
(8) **if** $(u > v)$ $u = (u - v)$ ≫2
(9) $x_1 = (x_1 - x_2)/4 \bmod p$
(10) **else** $v = (v - u) \gg 2$
(11) $x_2 = (x_2 - x_1)/4 \bmod p$
(12) **else if** $(u[1:0] == 2'b10)$
(13) **if** $((u >> 1) > v)$ $u = ((u$ ≫1) $- v)$ ≫1
(14) $x_1 = (x_1/2 - x_2)/2 \bmod p$
(15) **else** $u = u \gg 1$, $x_1 = x_1/2 \bmod p$
(16) $v = (v - (u \gg 1)) \gg 1$
(17) $x_2 = (x_2 - x_1/2)/2 \bmod p$
(18) **else if** $(v\ [1:0] == 2'b10)$
(19) **if** $(u > (v >> 1))$ $u = (u - (V$ ≫ 1)) ≫1
(20) $x_1 = (x_1 - x_2/2)/2 \bmod p$
(21) $v = v \gg 1$, $x_2 = x_2/2 \bmod p$
(22) else $v = ((v \gg 1) - u) \gg 1$
(23) $x_2 = (x_2/2 - x_1)/2 \bmod p$
(24) **else if** $(u \geq v)$
(25) $u = (u - v) \gg 1$, $x_1 = (x_1 - x_2)/2 \bmod p$
(26) else
(27) $v = (v - u) \gg 1$, $x_2 = (x_2 - x_1)/2 \bmod p$
(28) **end while**
(29) **return** $c = x_1$

Algorithm 3: Extended Euclidean modular inverse.

Montgomery point multiplication
Input: $k = (k_{l-1}, \ldots, k_0)$, point G
Output: $Q = kG$
(1) $R_0 = G$, $R_1 = 2G$, $i = l - 2$
(2) **while** ()
(3) **if** $(k_i == 0)$
(4) $R_1 = R_0 + R_1$, $R_0 = 2R_0$
(5) **else if** $(k_i == 1)$
(6) $R_0 = R_0 + R_1$, $R_1 = 2R_1$
(7) $i = i - 1$
(8) **end while**
(9) $Q = R_0$

Algorithm 4: Montgomery point multiplication.

In the data compression stage, dictionary matching stores the input *char* in the matching *window*, calculates the *hash* value according to the formula, reads the data from the hash index of the matching dictionary for short matching, and obtains the matching *length* and *offset*. The lengths are 1 byte and 2 bytes, respectively, and the matching window data are updated to the matching dictionary at the same time. This process is the data expansion stage. The data length changes from 1 byte to 4 bytes, and the *hash* calculation is as follows:

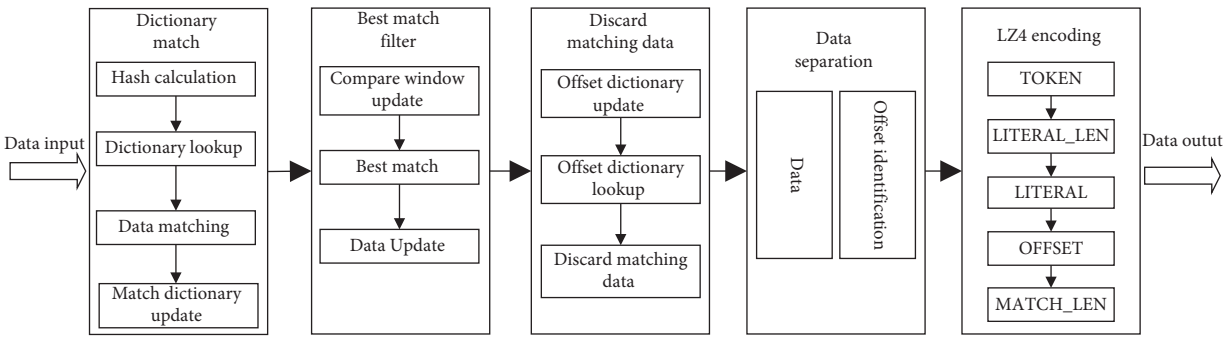

Figure 13: LZ4 algorithm processing structure.

$$hash = (\text{window}[0] \ll 4)\hat{}(\text{window}[1] \ll 3)\hat{}(\text{window}[2] \ll 3)\,\hat{\text{window}}[3]. \tag{12}$$

Optimal matching filtering processes the current data and neighboring data according to the optimal matching strategy and outputs them if they meet the conditions. Otherwise, it resets *length* and *offset* to 0, reducing the subsequent long matching operations. In the process of discarding matching data, firstly, the input data are stored in the offset dictionary. Then, the offset dictionary is matched according to the matching length. A maximum of 255 data points can be matched, and the matched data will be jumped to perform matching and discarding. Data separation divides the data into plaintext data *char* and offset identification according to the matching length. Offset identification includes the matching length, the matching offset, and the amount of plaintext data, which is also the basis for decompression.

The LZ4 encoding format jumps between the five states of token, literal length, literals, offset, and match length according to the offset identifier, and it outputs the final LZ4 data. The encoding structure is shown in Figure 14.

The LZ4 algorithm is a streaming algorithm that is used for storage-intensive, matching-intensive, and communication-intensive tasks. It needs to meet the requirements of calculation, storage, and communication at the same time. The data are transmitted in one direction in the data compression stage. Therefore, the algorithm is implemented in a pipelined computing mode. Data compression is divided into 11 subtasks. The same clock frequency is used in FPGA implementation. Under the clk trigger, 11 subtasks are calculated in parallel, and the data are transferred in sequence to form an 11-stage pipeline. The FPGA implementation of the LZ4 algorithm structure is shown below in Figure 15.

In the LZ4 encoding stage, the repeated data are removed, and this operation requires only a serial state machine. Even in the worst case, there is no data compression, however, only a few states are added for jumping, and the most time is still spent in the LITERAL state. One clock cycle processes one input. Therefore, the LZ4 algorithm uses an 11-stage pipeline to implement parallel calculations and uses serial calculations for encoding. The pipeline arithmetic module and the encoding module use FIFO to store communication data, preventing overflow and discarding the data generated by parallel calculations, which would result in incomplete data.

Multimodule parallelism is the simultaneous calculation of the same module in the same clock cycle using the control logic, and the performance can be doubled according to the number of parallel modules. Although the calculation logic of the LZ4 algorithm is complex, it occupies fewer resources. Therefore, according to the chip resources of FPGA, the data are distributed and recovered using the control of the state machine to realize multipath parallelism. The structure is shown in Figure 16.

The compressed data are transferred from the memory to the *data_fifo* inside the FPGA by the AXI bus protocol. The data are 256 bits wide, and the compressed data are 8 bits wide. Therefore, the data conversion module is added, the data are converted and stored in the FIFO buffer again, and the returned data still needs to be converted.

5. Experimental Results and Analysis

5.1. Experimental Environment. The main hardware computing component used in the scheme is the Zynq-XC7Z035 chip, which integrates a dual-core ARM A9 and a 275 K programmable logic unit and has hardware programming and software programming capabilities. The memory used is Micron's DDR3, with 1 GB of storage. There are Gigabit Ethernet ports on the ARM side and 10-Gigabit Ethernet ports and Gigabit Ethernet ports on the FPGA side. Flash is also included in the entire system. Writing *Bitstream* into Flash can complete the automatic loading and reconstruction of FPGA and provide detailed information about the main component configuration, as shown in Table 4.

The software environment for development is Vivado v2019.2, which is the supporting software of Xilinx and is used to perform the complete code writing, code simulation, compilation, placement and routing, downloading of bitstream files, and other operations.

The data protection scheme in this article is mainly for edge network equipment. Network video surveillance equipment plays an important role in public security, smart homes, smart cities, and other fields. The protection of data

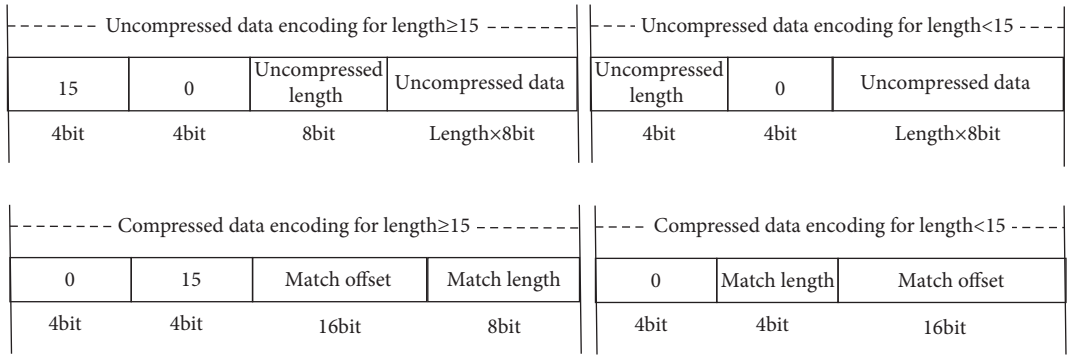

Figure 14: LZ4 coding structure.

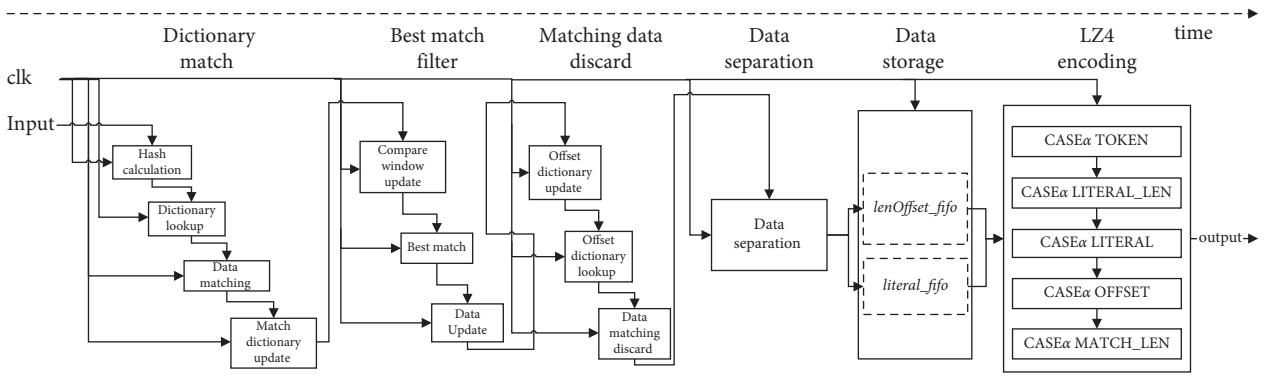

Figure 15: LZ4 algorithm FPGA implementation structure.

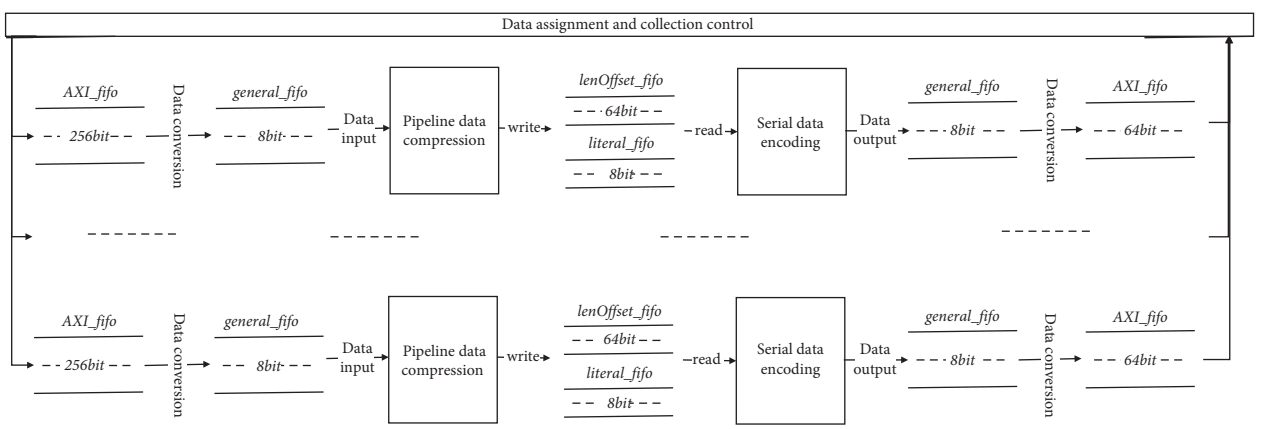

Figure 16: LZ4 multichannel parallel structure.

privacy is of great significance to safeguard people's lives and property. Therefore, we take video surveillance as an example to build a simulated test environment, and the real test example structure is shown in Figure 17.

The network monitoring device in Figure 17 has assigned IP addresses, which are 172.26.11.6 and 172.26.11.4, respectively. When the test is not performed, the terminal device can log in remotely to view the monitoring information. During the overall program performance test, the connection is made for the network monitor, router, Zynq, and terminal to initialize the protection program, including the initialization of the key and the initialization of the access control module. The protection equipment receives and processes the monitored real-time data from the network to verify whether the proposed solution meets the high-speed network data transmission requirements. In terms of security testing, simulated attackers are inserted into the front and back ends of the protection equipment, and real-time video surveillance information is obtained using network stealing methods. Based on the information obtained by the attackers, the security of the protection scheme is verified.

In this paper, the encryption algorithms SM3 and SM4 are deeply optimized. To verify the real performance, in the actual test, only the FPGA in Figure 17 needs to be statically reconstructed. The encryption algorithm is run on FPGA,

Table 4: Main component configuration information.

Component	Name	Configuration information
ARM	Model	Cortex-A9
	Architecture	ARMv7
	Frequency	800 M
FPGA	Logic cells	270 K
	Look up table (LUT)	171,900
	DSP slices	900
	Flip-flops	343,800
	GTP Transceiver	8 pairs of GTX, 10.315 Gbps, support PCIE Gen2
DDR	Memory size	1 G
	Main frequency	1600 MHz
	Maximum bandwidth	50 Gbps
Flash	Storage size	256 Mbit
	Maximum frequency	104 M

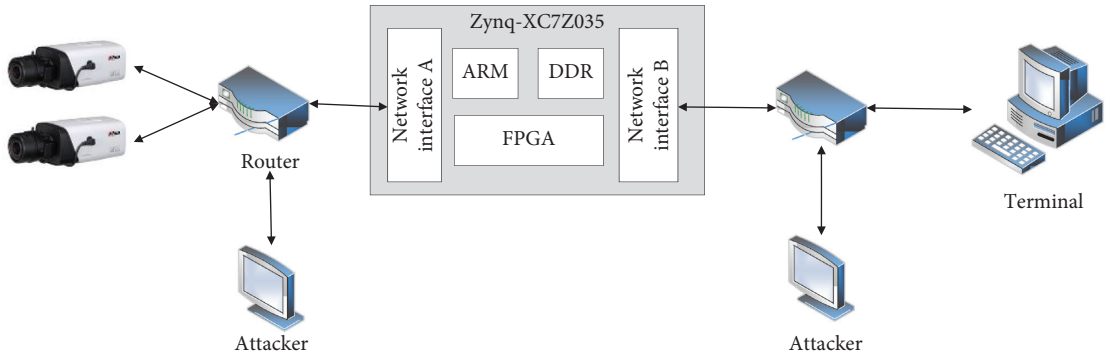

Figure 17: Real test environment structure.

and SM3 and SM4 are tested. The content is the real-time data of the monitoring equipment.

Encrypted data have no effect on the performance of the encryption algorithm, and the LZ4 compression algorithm is different. Therefore, in the actual test, not only the video content but also different content tests, including the text, data tables, pictures, and network mixed data packets, need to be tested. Thus, during FPGA testing, we transmit different data to Zynq through the network and perform multiple real tests on different content.

5.2. Experimental Results and Performance Analysis

5.2.1. Encryption Algorithm Implementation. The algorithm performance is based on throughput, and the calculation formula is as follows:

$$T = \frac{\mathrm{B} \times f_{\max} \times N}{d}, \tag{13}$$

where T is the throughput, B is the data block size, $f_{\max}$ is the maximum clock frequency of each scheme, N is the number of pipeline stages, and d is the calculation delay.

Three encryption algorithms are implemented on FPGA, among which the SM3 and SM4 algorithms are implemented using a full pipeline. Detailed information on the experimental results is shown in Table 5.

As the S-box in the SM4 algorithm uses block random access memory (BRAM), it occupies more BRAM. Many researchers have implemented parallel SM3 and SM4 algorithms on FPGAs, and the results are shown in Table 6.

Table 6 shows that after precomputation, pipeline technology, critical path optimization, and other methods are used to optimize the implementation, the performance of the SM3 and SM4 algorithms is improved by 43.2% and 36.1%, respectively, compared with other solutions, which has great advantages.

For the SM2 algorithm, the specific conditions of each mode's clock frequency, resource occupation, and calculation period are shown in Table 7.

Table 7 shows that when the point multiplication frequency is 29 MHz, the calculation can be completed after 3064 clock cycles, which is a very high calculation speed.

5.2.2. LZ4 Algorithm Implementation. The LZ4 algorithm reduces the amount of data transmitted on the network by compressing the network data. The single-module LZ4 and the two-way parallel LZ4 are implemented on FPGA. The results are shown in Table 8.

Table 8 shows that although the frequency of the implemented dual-channel LZ4 algorithm decreases, its performance increases by 1.84 times. According to different needs, the multichannel parallel LZ4 algorithm can be

Table 5: SM3 and SM4 algorithm implementation.

Algorithm	Algorithm structure	Frequency (MHz)	LUTs	FF	BRAM	Throughput (Gbps)
SM3	65-stage pipeline	240	24774	33189	0	115.2
SM4	98-stage pipeline	335	8284	10627	128	42.88

Table 6: Algorithm implementation comparison.

Algorithm	Implementation	Frequency (MHz)	Throughput
SM3	Ours	240	115.2 Gbps
	Zheng et al. [50]	36	263 Mbps
	Zang et al. [51]	415	6.4 Gbps
SM4	Ours	335	42.88 Gbps
	Fan et al. [34]	250	528 Mbps
	Yang et al. [35]	250	31.5 Gbps

Table 7: Implementation of SM2 modules.

Module	Frequency (MHz)	LUT	FF	DSP	Calculation period
Point addition	29	8911	4442	144	12
Point doubling	29	10056	3824	144	12
Point multiplication Control	29	1499	7251	0	3064
Coordinate transformation	29	16927	4356	144	225

Table 8: LZ4 algorithm implementation results.

Algorithm	Frequency (MHz)	LUT	FF	BRAM	Throughput (Mbps)
Single-module LZ4	110	8583	8072	98	880
Dual parallel LZ4	101	17481	16,632	196	1616

configured and transplanted to different application scenarios.

Different types of data are tested to verify the actual performance of the LZ4 algorithm. The results are shown in Table 9.

Table 9 shows that the LZ4 algorithm has the highest compression rate for pictures, i.e., the worst effect. It is because the LZ4 algorithm is a general-purpose compression algorithm, and an image requires a dedicated compression algorithm to reduce the compression rate. LZ4 has a better compression effect for other types of data.

The LZ4 algorithm is a solution proposed in this paper to reduce the network load. Different types and sizes of data are input for processing within a period of time, and the output data volume is determined. The test results are shown in Figure 18.

The experimental results in Figure 18 show that the compression performance of the LZ4 compression algorithm is related to the compressed data. By calculating the average compression ratio, it can be concluded that the LZ4 algorithm reduces the amount of original network data by 10%, which shows that the scheme proposed in this paper is effective.

5.2.3. Implementation of FPGA Underlying the Related Modules. The encryption algorithm and data compression algorithm are the core parts of the scheme of this article. To realize a complete data processing scheme, 10-Gigabit network interfaces, Gigabit network interfaces, data packet analysis and encapsulation, and key generation modules implemented on FPGA are needed. The realization of each functional module is shown in Table 10.

The processing flows of the Gigabit and 10-Gigabit networks are basically the same, except that the working frequencies are different, and the bit widths of the processed data are 1 byte and 8 bytes, respectively. Therefore, bit width conversion is required before a data packet is encrypted or decrypted.

5.2.4. Network Performance Test. In a Gigabit network, since the LZ4 algorithm processes only the data filtered by ACL, a single-channel LZ4 algorithm can meet the requirements. Among the modules implemented, the lowest frequency of the LZ4 algorithm is 110 MHz, which gives the lowest throughput. Therefore, when implemented, the SM3 and SM4 algorithms also work at 110 MHz, which can meet the performance requirements. The network data of different sizes are encrypted, and the network performance is shown in Table 11.

Table 11 also shows that the data delay gap of different data sizes is relatively small, i.e., approximately 480. It is because the key agreement and key encryption calculation are first carried out in the entire system, which consumes some of the time. When the data start entering the compression and encryption stage, the data are processed continuously, which is independent of the size of the data. The processing time is from the beginning of data input to the time at which the output is complete. The greater the data length, the longer the processing time. As the data length

Table 9: LZ4 compression rate and compression speed.

Type of data	Before compression	After compression	Compression rate (%)	Compression speed (MB/s)
Network packet	1,536,000	1,406,664	91.58	107
Image	430,080	423,154	98.39	106
Text	147,456	680,686	46.16	109
Data sheet	16,461	13,379	81.28	108.4

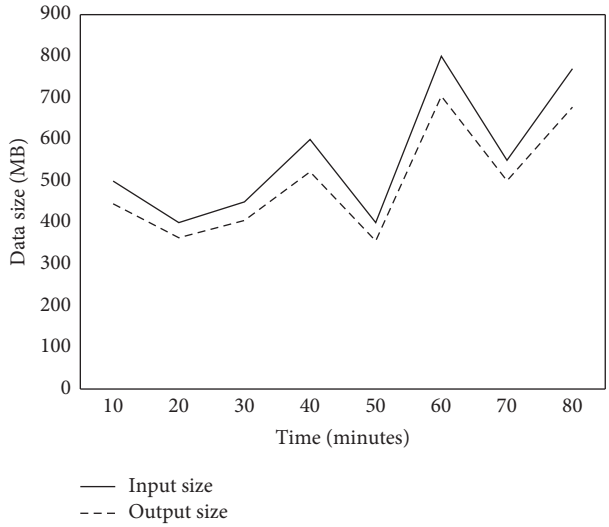

Figure 18: Network load changes.

Table 10: FPGA-related module implementation.

Functional module	Description	Frequency (MHz)	LUT	FF
ARM	ARM communication module	100	6487	8296
SFP_10GE_MAC	10-Gigabit network interface	156.25	4816	5457
Tri_Mode_Ethernet_MAC	Gigabit network interface	125	637	879
DDR_WR	Memory read-write interface	666.67	9321	7549
Frame_Parse	Data frame analysis	158	900	2336
Frame_Filter	Data frame filtering	158	1026	2673
Frame_Package	Data frame encapsulation	158	274	476
Key_LFSR	Key generation and update	158	517	1016
DataTransmission	High-speed data transmission	158	735	2037

Table 11: Gigabit network performance test.

Data length (MB)	Output delay (μs)	Processing time (s)
100	480	1.7
200	478	2.3
300	484	3.05
400	481	3.78
500	484	4.17
600	483	5.15
700	476	6.11
800	488	6.504

increases, the processing performance of network data is closer to 1000 Mb/s, reaching 987 Mb/s. It indicates that when data protection is added, there is no additional delay in network transmission, which meets the network requirement of a low delay.

In the 10-Gigabit network, the data input speed is 10 Gb/s, and the 4-channel parallel LZ4 algorithm is implemented on FPGA with a frequency of 100 MHz. The network performance test is shown in Table 12, and the network performance can be calculated as reaching 9302 Mb/s.

The network performance of the Gigabit network and the 10-Gigabit network reached 987 Mb/s and 9302 Mb/s, respectively, fulfilling the data processing requirements of Gigabit and 10-Gigabit networks in the overall design scheme. The encryption of the network data guarantees the privacy of the data, the effectiveness of the network load, and little network delay.

5.2.5. Computational Time Complexity Analysis. In the data protection scheme designed in this paper, the main calculation modules are key generation, high-speed data transmission, access control, the symmetric encryption algorithm SM4, the hash algorithm SM3, the key agreement algorithm SM2, and the data compression algorithm.

Key generation includes two parts: LFSR pseudo-random number generation and scrambling factor generation.

Table 12: 10 Gb network performance test.

Data length (MB)	Output delay (μs)	Processing time (s)
10,000	482	9.8
20,000	484	17.1
30,000	477	25
40,000	483	33.4
50,000	482	43

LFSR generates 128-bit data. Hence, 128-level LFSR is required, which consumes 128 clock cycles. The scrambling factors FID, temperature, and RS are calculated at the same time during the initial configuration of the key. After the pseudo-random sequence is generated, the key generation is completed by one clock consumption. The calculation time T_{Key} for key generation is 129 clocks consumed.

High-speed data transmission is mainly used for data transmission between DDR and FPGA. It adopts burst data transmission and is actually tested on FPGA. When the frequency is 158 MHz, the maximum throughput is 4157 MB/s. When the size of the transmitted data is $\text{Size}_{\text{Data}}$, the transmission time is $\text{Size}_{\text{Data}}/4157$.

Access control includes data frame analysis, filtering, and encapsulation. When implementing, the corresponding data are extracted from a clock using the state selector and filtered through multicondition matching, and the corresponding operations are performed. Finally, the corresponding data are filled into the fixed position of the data frame. The computational complexity is mainly the realization of the state selector, the final frequency of the hardware realization is 158 MHz, and each data frame only needs to consume 1 clock.

SM4, SM3, and LZ4 are the most computationally complex parts of the data protection scheme. This article adopts a variety of optimization schemes to focus on implementation. A set of data can be calculated on each clock. Therefore, SM4, SM3, and LZ4 calculate one set for each clock. The data sizes are 128 bits, 480 bits, and 8 bits.

SM2 encrypts the key. It is calculated only once in a period of time, and the amount of calculation is small. Hence, the area is optimized. The time for one calculation is mainly point multiplication, and the time consumed by T_{SM2} is 3064 clock cycles.

In the data protection scheme, using the implementation and testing of each module, the calculation time complexity of the key factor in the LZ4 algorithm performance scheme is the key factor in the calculation time complexity. Therefore, when N groups of data need to be protected and the size of each group of data is M bytes, the total calculation time T_{all} is given by the following equation:

$$T_{\text{all}} = T_{\text{Key}} + T_{\text{SM2}} + \left(T_{\text{Delay\_LZ4}} + N \times M\right) + T_{\text{Delay\_SM4}} + T_{\text{Delay\_SM3}}. \tag{14}$$

$T_{\text{Delay\_LZ4}}$, $T_{\text{Delay\_SM4}}$, and $T_{\text{Delay\_SM3}}$ are the output delays of LZ4, SM4, and SM3, respectively, and the specific values are 13, 99, and 66. LZ4 processes 1 byte of data each time and takes a total of $N{\times}M$. As SM4, SM3, and LZ4 are calculated in parallel, when the last set of data is compressed, there is only one set of data to be encrypted, and the processing of N sets of data is completed after the encryption is delayed and output. The ACL and the data transmission module have small delays and are parallel modules. Therefore, the calculation time complexity is mainly based on data compression and encryption. When the data size is 1510, the time consumed can be calculated as shown in the following formula:

$$T_{\text{all}} = 129 + 3064 + (13 + N \times 1510) + 99 + 66. \tag{15}$$

5.3. Safety Analysis

5.3.1. Key Randomness Analysis. Encryption keys are the key to ensuring data security. In this paper, while incorporating the idea of mimic defense, three kinds of LFSR, *DF,* and time t are used as variable factors to generate keys. The initial key generated at time t is Key $(t) = \text{LFSR} \oplus \text{DF}$, $\text{LFSR} \in \{\text{LFSR}_1, \text{LFSR}_2, \text{LFSR}_3\}$, $\text{DF} \in \{FID, temperature, RS\}$.

According to different configuration attributes, the initial key has 9 different configuration combinations, and the LFSR and the scrambling factor are dynamically variable and related to the operating state of the system at the time. Therefore, the probability of generating the same initial key is low, ensuring the initial unpredictability of the key, and the encryption key of different frame data *fdata* implements the idea of "one frame, one key." The calculation of the key within time t is as follows:

$$\text{key}_{\text{FID}} = \begin{cases} \text{Key}(t), & \text{FID} = 1, \\ \text{key}_{\text{FID}-1} \oplus \text{SM3}(fdata), & \text{FID} > 1. \end{cases} \tag{16}$$

The edge network and the server agree to generate an initial key at different time intervals using key agreement, and the minimum value of the time interval must meet the calculation time of the SM2 encryption key. The key is constantly changing, and the value at a certain moment is different from that at other moments, i.e., Key $(t_1) \neq$ Key $(t_2) \neq$, ..., Key (t_l), to ensure the dynamic nature of the key transformation. Different LFSRs have different rules for generating keys, which increases the attack disturbance, making it difficult for an attacker to crack the key in a short time, and the diversity of disturbance factors increases the difficulty of the attack. The pseudo-random number generator itself has a certain degree of randomness. The multiple initial keys generated in a certain period of time are again randomly selected as the final key, which is the second random selection, thereby further increasing the encryption key randomness.

5.3.2. Security Analysis of the Data Protection Scheme. The cryptographic standards proposed in China have high complexity and security, which makes the cost of deciphering by attackers exceed the possible benefits.

The encryption key uses the 256-bit SM2 algorithm to complete key negotiation between the two parties. It is based on the problem of discrete logarithms and has relatively high security and resistance to attacks. According to the

Table 13: Comparison of the security solutions.

Scheme	Year	Technology/method	Computing platform	Goals
Dar et al. [25]	2020	Context-Aware, RSA, ECC, DSA, AES, Blowfish, RC6	CPU	Protect IoT data security and reduce execution time and energy consumption
Zhao et al. [28]	2020	Data aggregation, bilinear pairing, Paillier homomorphic encryption	CPU	Solve the problem of limited bandwidth and data privacy protection of a single device
Soliman et al. [30]	2019	AES, LFSR, COLM, OCB	ARM + FPGA	Add a new security dimension to IoT devices and reduce area utilization and power consumption
Yang et al. [35]	2019	SM4-XTS, SM2, modular exponentiation	FPGA	Meet high-concurrency big data security requirements
Wang et al. [36]	2021	Double colour images encryption, compressive sensing	CPU	Reduce data volume and improve the efficiency of transmitting data
Fouad et al. [37]	2021	RSA, hybrid compression	CPU/ARM	Reduce the physical footprint and protect encrypted hidden data
Our System	—	Access control list One frame, one key SM4, SM3, SM2, LZ4	ARM + FPGA	Protect the privacy of edge network data, and increase the amount of data transmission

published E (Fp (a, b)), base point G, and order n, $2G$, $3G$, . . ., nG can be calculated, and $nG = O$. When a large number k is given, $P = kG$ can be easily calculated. However, given P and G, it is very difficult to infer k.

The SM3 algorithm is irreversible, and the hash values obtained differ for different content. Any change in the input information will cause a significant change in the hash result. It ensures that the key parameters of different data frames are very different, thereby protecting the frame data and preventing attackers from inverting the key based on the content of the frame. The SM3 operation is anticollision, and the collision threshold is on the order of 2^{256}. It is difficult to find two pieces of information with the same hash result, which can effectively prevent differential attacks.

Similarly, the SM4 algorithm uses brute-force cracking that requires an order of magnitude of 2^{128}. At the same time, data encryption uses the "one frame, one password" scheme. Even if a key replay attack or a ciphertext attack is used to crack a set of data frames, it is necessary to re-establish the attack chain and crack other groups of data.

The LZ4 algorithm provides data compression, which not only increases the network data load but also plays a role in data protection. Assume that when an attacker intercepts all encrypted data, he or she obtains a certain frame of plaintext data through key exhaustion attacks, ciphertext-only attacks, and differential attacks. The time required is t. If the key information required by the attacker is in the last frame of the data and the total data frame is m, when the data are not compressed, it takes only time t to crack, and after data compression is added, all the data need to be decrypted. The time is mt, and decrypting only the current frame data may result in garbled information i.e., passing the first SM4 encryption line of defense and the second line of defense brought by data compression so that the attacker's attack chain is expanded from part to all of the data increases the cost of the attack, thereby further ensuring the privacy of all data.

This article uses video surveillance to test and add an attacker before and after the protection scheme equipment. At the front end without protection, the surveillance video can be obtained directly. At the back end of the protection equipment, the data obtained by the attacker are worthless garbled data because of encryption, and only after the terminal is decrypted can the correct data be obtained. Therefore, applying the solution proposed in this article to edge network equipment can protect the confidentiality of data and achieve a higher security algorithm, which increases the attack difficulty.

5.3.3. Comparison with Other Security Solutions. The lightweight data protection scheme designed in this paper is oriented toward edge networks, and mobile networks and edge sensor networks are important components of edge networks. Therefore, the design scheme in this paper is compared with related security mechanisms, as shown in Table 13:

In terms of computing platforms, CPUs are used for implementation in the literature [25, 28, 36, 37]. Even if the calculation is performed using parallel technology, such as multithreading, it is still difficult for the computing efficiency to meet high-speed network data processing requirements, and FPGA using register transfer-level pipeline technology can achieve higher performance and has a larger performance advantage compared with CPU. In previous work [28, 30] using an FPGA for implementation in high-performance network applications, the encryption scheme does not consider the key factor, and the completeness of the system does not consider the changeable network environment. In terms of the core algorithm, the RSA algorithm is used for encryption in the literature [25, 37]. Under the same encryption length, the security of SM2 is much higher than that of RSA. At the same time, the calculation of multiple algorithms of the same type will increase the complexity of the algorithm and consume calculation resources. Encryption and compression are used to protect the security of image data in the literature [36, 37], however, computational efficiency and applicability are lacking for other types of data.

Table 13 shows that this article combines 4 security strategies to achieve data security in the edge network. Compared with other solutions, the proposed approach

provides more comprehensive data protection. The first strategy is the access control list, which is used to filter illegal data and reduce the recurrence of illegal data. The transmission and direct forwarding of unimportant data reduce processing, thereby reducing the energy consumption of system operations. Secondly, the use of a "one frame, one key" security key strategy resists ciphertext-only attacks and destroys the construction of the attack chain. Then, data compression not only reduces the space occupied by the data but also plays an important role in data security. Finally, the data encryption algorithm contains different types of operations that can perform key encryption, data encryption, and key update functions to meet different needs.

The design scheme of this article uses ARM and FPGA to realize the security mechanism together and implements a high-performance encryption algorithm and data compression algorithm on FPGA. FPGA is resistant to interference, and it is difficult for outsiders to obtain authorization to make internal changes. The security of the system is protected on the hardware. ARM can detect the operating state of FPGA using the feedback of FPGA, thereby changing the operating state of FPGA in real time and flexibly performing configuration and calculations. The algorithm implemented in this paper occupies fewer FPGA resources and has low operating power consumption. ARM is mainly used for mobile devices. It has the characteristics of a small structural area and high operational flexibility. Therefore, the combination of ARM and FPGA in an edge network device can satisfy the lightweight edge data protection requirements of the network.

6. Conclusions

The edge of the network is the first line of defense against malicious attacks. The lightweight edge network data protection scheme proposed in this paper is mainly composed of encryption algorithms and data compression algorithms. It uses LFSR and disturbance factors to generate a random, dynamic, and diverse initial key. The key is updated with the hash value of the data frame to realize the encryption method of "one frame, one key," and the whole system is constructed using the heterogeneous calculation method of ARM-FPGA. The experimental results and analysis show that the encryption system has not only higher encryption throughput but also higher security. It can effectively prevent data leakage and resist key exhaustive attacks and ciphertext attacks. At the same time, the added data compression module not only reduces the amount of network data but also displays compressed, garbled information when the attacker obtains part of the data. It also provides data privacy protection to a certain extent. The lightweight data compression algorithm LZ4 can increase the network load by 10%.

The security protection mechanism proposed in this article includes access control, random key generation, data compression, and data encryption modules. Different modules can be dynamically transformed according to actual needs. For example, the efficient optimization method of the encryption algorithm on FPGA in this article is also applicable to the advanced encryption standard (AES), elliptic curve cryptography (ECC), and SHA-256, which are equivalent algorithms. Hence, they can be replaced flexibly as needed. On the other hand, with the rapid development of chip technology, mobile devices integrate ARMs, GPUs, FPGAs, etc., to form a heterogeneous computing platform. As part of the edge network, the lightweight data security solution in this article is also applicable to mobile devices. It protects the data security of mobile devices and improves the data transmission speed through data compression.

In future work, we will conduct more in-depth research on edge network security mechanisms. The first objective is the division of data security levels. There are differences in the security level and implementation complexity of different encryption algorithms. Achieving the most reconfigurable security protection for optimal planning of different data guarantees maximum utilization of resources and energy consumption. The second goal is to further establish the security mechanism of the edge network using reliability theory, network coding, combination design, etc., develop the theoretical model of the edge network, and evaluate the security of the data protection mechanism based on theoretical simulations. Finally, the edge node and the server establish a dedicated and lighter-weight secure communication protocol. Therefore, it is important to ensure that when an FPGA-based edge device is attacked, it can actively discover, dynamically reconfigure, and improve the device's antiattack ability.

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 paper.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 61702518 and in part by the National Key R&D Program Key Special Project of China under Grant 2018XXXXXXXX01.

References

[1] J. Zhou, H. J. Shen, Z. Y. Lin, Z. F. Cao, and X. L. Dong, "Research advances on privacy preserving in edge computing," *Journal of Computer Research and Development*, vol. 57, no. 10, pp. 2027–2051, 2020.

[2] A. Paul, *2020 Internet Crime Report*, Internet Crime Complaint Center, USA, 2020.

[3] W. House, *Trustworthy Cyberspace: Strategic Plan for the Federal Cybersecurity Research and Development Program. Report of the National Science and Technology Council*, Executive Office of the President, 2011.

[4] S. Tennina, O. Gaddour, A. Koubâa, F. Royo, M. Alves, and M. Abid, "Z-Monitor: a protocol analyzer for IEEE 802.15.4-

based low-power wireless networks," *Computer Networks*, vol. 95, pp. 77–96, 2016.

[5] Y. Zou and G. Wang, "Intercept behavior analysis of industrial wireless sensor networks in the presence of eavesdropping attack," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 2, pp. 780–787, 2016.

[6] M. Albanese, S. Jajodia, and S. Venkatesan, "Defending from stealthy botnets using moving target defenses," *IEEE Security & Privacy*, vol. 16, no. 1, pp. 92–97, 2018.

[7] S. Venkatesan, M. Albanese, and K. Amin, "A moving target defense approach to mitigate DDoS attacks against proxy-based architectures," in *Proceedings of the 2016 IEEE Conference on Communications and Network Security (CNS)*, pp. 198–206, IEEE, Philadelphia, PA, USA, October 2016.

[8] H. Almohri, L. T. Watson, and D. Evans, "Predictability of IP address allocations for cloud computing platforms," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 500–511, 2019.

[9] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "An effective address mutation approach for disrupting reconnaissance attacks," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 12, pp. 2562–2577, 2015.

[10] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Adversary-aware IP address randomization for proactive agility against sophisticated attackers," in *Proceedings of the 2015 IEEE Conference on Computer Communications (INFOCOM)*, pp. 738–746, IEEE, Kowloon, Hong Kong, April 2015.

[11] F. Song, Y. T. Zhou, Y. Wang, T. Zhao, I. You, and H. K. Zhang, "Smart collaborative distribution for privacy enhancement in moving target defense," *Information Sciences*, vol. 479, pp. 593–606, 2018.

[12] H. Tang, Q. T. Sun, X. Yang, and K. Long, "A network coding and DES based dynamic encryption scheme for moving target defense," *IEEE Access*, p. 1, 2018.

[13] D. Ma, L. Wang, and L. Cheng, "Thwart eavesdropping attacks on network communication based on moving target defense," in *Proceedings of the 2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*, pp. 1-2, IEEE, Las Vegas, NV, USA, December 2016.

[14] G. Lin, M. Dong, and K. Ota, "Security function virtualization based moving target defense of SDN-enabled smart grid," in *Proceedings of the 2019 IEEE International Conference on Communications (ICC) (ICC 2019)*, IEEE, Shanghai, China, May 2019.

[15] S. Wang, Y. Zhou, and R. Guo, "A novel route randomization approach for moving target defense," in *Proceedings of the 2018 IEEE 18th International Conference on Communication Technology (ICCT)*, IEEE, Chongqing, China, October 2018.

[16] E. M. Ghourab and M. Azab, "Benign false-data injection as a moving-target defense to secure mobile wireless communications," *Ad Hoc Networks*, vol. 102, no. 9, Article ID 102064, 2020.

[17] H. Hu, J. Wu, Z. Wang, and G. Cheng, "Mimic defense: a designed-in cybersecurity defense framework," *IET Information Security*, vol. 12, no. 3, pp. 226–237, 2018.

[18] Y. Ming and E. Wang, "Identity-based encryption with filtered equality test for smart city applications," *Sensors*, vol. 19, no. 14, p. 3046, 2019.

[19] X. Zhou, B. Li, Y. Qi, and W. Dong, "Mimic encryption box for network multimedia data security," *Security and Communication Networks*, vol. 2020, no. 2, pp. 1–24, Article ID 8868672, 2020.

[20] N. M. Truong, M. Aoki, Y. Igarashi, M. Saito, and K. Yamamoto, "Real-time lossless compression of waveforms using an FPGA," *IEEE Transactions on Nuclear Science*, vol. 65, no. 9, pp. 2650–2656, 2018.

[21] C. Wang, D. Wang, Y. Tu, G. Xu, and H. Wang, "Understanding node capture attacks in user authentication schemes for wireless sensor networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 507–523, 2020.

[22] Q. Tong, Z. Zhang, W. Zhang, and J. Wu, "Design and implementation of mimic defense web server," *Journal of Software*, vol. 28, no. 4, pp. 883–897, 2017.

[23] P. Pavithran, S. Mathew, S. Namasudra, and P. Lorenz, "A novel cryptosystem based on DNA cryptography and randomly generated mealy machine," *Computers & Security*, vol. 104, 2020.

[24] P. Zhang, C. Lin, Y. Jiang, Y. Fan, and X. Shen, "A lightweight encryption scheme for network-coded mobile ad hoc networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 9, pp. 2211–2221, 2014.

[25] Z. Dar, A. Ahmad, F. A. Khan et al., "A context-aware encryption protocol suite for edge computing-based IoT devices," *The Journal of Supercomputing*, vol. 76, no. 4, pp. 2548–2567, 2020.

[26] S. Qiu, D. Wang, G. Xu, and S. Kumari, "Practical and provably secure three-factor Authentication protocol based on extended chaotic-maps for mobile lightweight devices," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[27] C. Y. Wang, D. Wang, G. A. Xu, and D. B. He, "Efficient privacy-preserving user authentication scheme with forward secrecy for Industry 4.0," *Science China Information Sciences*, 2020.

[28] O. Zhao, X. Liu, X. Li, P. Singh, and F. Wu, "Privacy-preserving data aggregation scheme for edge computing supported vehicular ad hoc networks," *Transactions on Emerging Telecommunications Technologies*, 2020.

[29] A. L. P. de França, R. P. Jasinski, and V. A. Pedroni, "Moving network protection from software to hardware: an energy efficiency analysis,"*in Proceedings of the 2014 IEEE ComputerSociety Annual Symposium on VLSI (ISVLSI)*, IEEE, Tampa, FL, USA, pp. 456–461, 2014.

[30] S. Soliman, M. A. Jaela, A. M. Abotaleb et al., "FPGA implementation of dynamically reconfigurable IoT security module using algorithm hopping," *Integration*, vol. 68, pp. 108–121, 2019.

[31] S. Pontarelli, C. Greco, E. Nobile, and S. Teofili, "Exploiting dynamic reconfiguration for FPGA based network intrusion detection systems," in *Proceedings of the 2010 International Conference on Field Programmable Logic and Applications (FPL)*, pp. 10–14, IEEE, Milano, Italy, August 2010.

[32] S. Pontarelli, G. Bianchi, and S. Teofili, "Traffic-aware design of a high-speed FPGA network intrusion detection system," *IEEE Transactions on Computers*, vol. 62, no. 11, pp. 2322–2334, 2013.

[33] Y. Zhang, D. He, M. Zhang, and K.-K. Raymond Choo, "A provable-secure and practical two-party distributed signing protocol for SM2 signature algorithm," *Frontiers of Computer Science*, vol. 14, no. 3, 2019.

[34] L. Y. Fan, M. Zhou, J. J. Luo, and H. L. Liu, "IC design with multiple engines running CBC mode SM4 algorithm," *Journal of Computer Research and Development*, vol. 55, no. 6, pp. 1247–1253, 2018.

[35] G. Q. Yang, H. C. Ding, J. Zou, H. Jiang, and Y. Q. Chen, "A big data security scheme based on high-performance cryptography implementation," *Journal of Computer Research and Development*, vol. 33, pp. 12755–12776, 2019.

[36] K. Wang, X. Wu, and T. Gao, "Double color images compression–encryption via compressive sensing," *Neural Computing and Applications*, pp. 1–22, 2021.

[37] O. Fouad, A. Khalaf, A. I. Hussein, and H. F. A. Hamed, "Hiding data using efficient combination of RSA cryptography, and compression steganography techniques," *IEEE Access*, vol. 9, pp. 31805–31815, 2021.

[38] Z. Zhang, X. Q. Zhang, D. C. Zuo, and G. D. Fu, "Research on target tracking application deployment strategy for edge computing," *Journal of Software*, vol. 31, no. 9, pp. 2691–2708, 2020.

[39] A. K. Panda, P. Rajput, and B. Shukla, "Design of multi bit LFSR PNRG and performance comparison on FPGA using VHDL," *International Journal of Advances in Engineering & Technology*, vol. 3, no. 1, pp. 566–571, 2012.

[40] Y. Liu, H. Liang, W. Wang, M. Wang, and J. Díaz-Verdejo, "New linear cryptanalysis of Chinese commercial block cipher standard SM4," *Security and Communication Networks*, vol. 2017, Article ID 1461520, 2017.

[41] A. P. Kakarountas, H. Michail, A. Milidonis, C. E. Goutis, and G. Theodoridis, "High-speed FPGA implementation of secure hash algorithm for IPSec and VPN applications," *The Journal of Supercomputing*, vol. 37, no. 2, pp. 179–195, 2006.

[42] B. S. Jangareddi and G. Sridevi, "A cryptographic based implementation of secure hash algorithm by using microblaze processor," *International Journal of Research in Computer and Communication Technology*, vol. 3, no. 10, pp. 1379–1383, 2014.

[43] D. Ravilla and C. S. R. Putta, "Enhancing the security of MANETs using hash algorithms," *Procedia Computer Science*, vol. 54, pp. 196–206, 2015.

[44] J. Zou and L. Dong, "Improved preimage and pseudo-collision attacks on SM3 hash function," *Journal on Communications (in Chinese)*, vol. 39, no. 1, pp. 46–55, 2018.

[45] M. S. Hossain, Y. Kong, E. Saeedi, and N. C. Vayalil, "High-performance elliptic curve cryptography processor over NIST prime fields," *IET Computers & Digital Techniques*, vol. 11, no. 1, pp. 33–42, 2016.

[46] S. Khan, K. Javeed, and Y. A. Shah, "High-speed FPGA implementation of full-word Montgomery multiplier for ECC applications," *Microprocessors and Microsystems*, vol. 62, pp. 91–101, 2018.

[47] W. Yu, K. Wang, B. Li, and S. Tian, "Montgomery algorithm over a prime field," *Chinese Journal of Electronics*, vol. 28, no. 1, pp. 43–48, 2019.

[48] K. Javeed and X. Wang, "FPGA based high speed SPA resistant elliptic curve scalar multiplier architecture," *International Journal of Reconfigurable Computing*, vol. 2016, Article ID 6371403, 10 pages, 2016.

[49] J. Kim and J. Cho, "Hardware-accelerated fast lossless compression based on LZ4 algorithm, International academy of computing technology (IACT)," in *Proceedings of the 2019 3rd International Conference on Digital Signal Processing (ICDSP 2019)*, pp. 67–70, Jeju Island, Republic of Korea, February 2019.

[50] X. Zheng, X. Hu, J. Zhang, J. Yang, S. Cai, and X. Xiong, "An efficient and low-power design of the SM3 hash algorithm for IoT," *Electronics*, vol. 8, no. 9, p. 1033, 2019.

[51] S. Zang, D. Zhao, Y. Hu et al., "A high speed SM3 algorithm implementation for security chip," in *Proceedings of the 2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pp. 915–919, Chongqing, China, March 2021.