


Article

GFRX: A New Lightweight Block Cipher for Resource-Constrained IoT Nodes

Xing Zhang ¹, Shaoyu Tang ¹, Tianning Li ², Xiaowei Li ¹ and Changda Wang ^{1,*}¹ School of Computer Science & Communication Engineering, Jiangsu University, Zhenjiang 212013, China² School of Electrical & Information Engineering, Jiangsu University, Zhenjiang 212013, China

* Correspondence: changda@ujs.edu.cn

Abstract: The study of lightweight block ciphers has been a “hot topic”. As one of the main structures of block ciphers, the Feistel structure has attracted much attention. However, the traditional Feistel structure cipher changes only half of the plaintext in an iterative round, resulting in slow diffusion. Therefore, more encryption rounds are required to ensure security. To address this issue, a new algorithm, GFRX, is proposed, which combines a generalized Feistel structure and ARX (Addition or AND, Rotation, XOR). The GFRX algorithm uses an ARX structure with different non-linear components to deal with all the branches of a generalized Feistel structure so that it can achieve a better diffusion effect in fewer rounds. The results of a security analysis of the GFRX algorithm show that the effective differential attacks do not exceed 19 rounds and that the effective linear attacks do not exceed 13 rounds. Therefore, the GFRX algorithm has an adequate security level for differential and linear analysis. Avalanche test results obtained for the GFRX algorithm show that the GFRX algorithm has strong diffusion and only takes six rounds to meet the avalanche effect. In addition, the GFRX algorithm can achieve different serialization levels depending on different hardware resource requirements and can achieve full serialization, which ensures operational flexibility in resource-constrained environments.

Keywords: lightweight block cipher; generalized Feistel structure; ARX; serialization implementation



Citation: Zhang, X.; Tang, S.; Li, T.; Li, X.; Wang, C. GFRX: A New Lightweight Block Cipher for Resource-Constrained IoT Nodes. *Electronics* **2023**, *12*, 405. <https://doi.org/10.3390/electronics12020405>

Academic Editor: Elif Bilge Kavun

Received: 30 November 2022

Revised: 7 January 2023

Accepted: 10 January 2023

Published: 12 January 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

With the development of the Internet of Things (IoT), the number of IoT devices has increased very rapidly. Information interaction based on identification technology, pervasive computing and edge computing [1] is the core of the IoT, in which IoT nodes are the medium of information interaction. IoT nodes collect data by sensing external changes and sending the collected data to a data analysis center. At present, IoT nodes, such as RFID tags and sensors, are widely used in manufacturing, smart cities, defence and military applications, public security and other fields. Generally, data collected by IoT nodes are highly confidential; hence, it is particularly important to ensure the security of data transmission and storage [2–4]. Therefore, with respect to the special requirements of the IoT, a dedicated encryption algorithm is urgently needed for data storage and transmission in the perception layer of the IoT.

Common IoT nodes consist of various sensors, such as pressure sensors, temperature sensors, humidity sensors and smoke sensors. The sensors perceive external changes and generate analog signals. To prevent data leakage or tampering, it is necessary to deploy a lightweight encryption chip inside the sensor node. When the sensor generates an analog signal, a compatible A/D converter is needed to convert the analog signal into digital form as input to an encrypted chip. Based on the cryptographic functions coded inside the chip, the sensed data is encrypted and then transmitted to the cloud server through a data-analysis center for storage. Authorized users can only access the encrypted data subsequently from the cloud server through the cloud service provider.

The specific process is shown in Figure 1. However, IoT nodes are usually small embedded devices, which have relatively limited computing power and storage space. It is noteworthy that traditional encryption algorithms, such as AES (Advanced Encryption Standard) [5] and IDEA (International Data Encryption Algorithm) [6], are not suitable for resource-constrained devices. Therefore, a lightweight cryptographic algorithm is essential for IoT nodes.

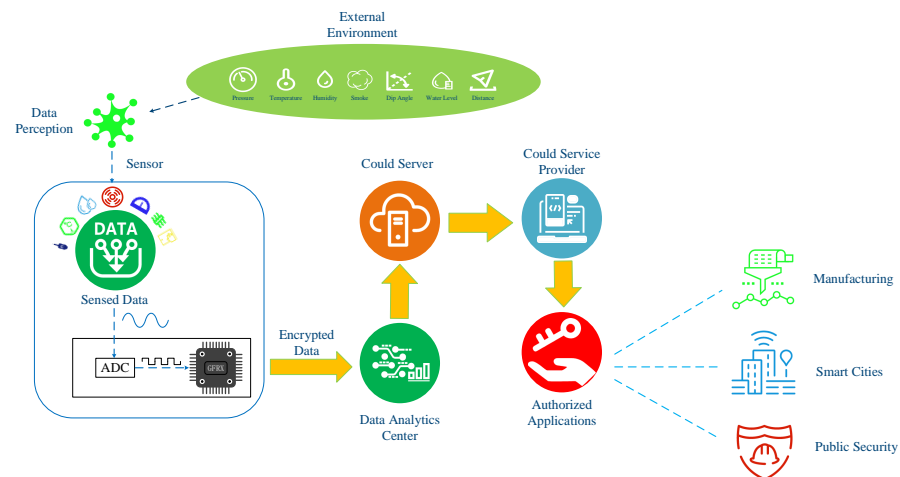


Figure 1. The process of sensor information acquisition.

In recent years, many excellent lightweight cryptographic algorithms have emerged. These algorithms can be roughly divided into four types of structures [7]: SPN (substitution permutation networks), Feistel, ARX and a mixture of these. Lightweight block cipher algorithms, such as PRESENT [8], GIFT [9], Loong [10] and ILEA [11], are all based on the SPN structure. The SPN structure usually uses S-boxes as the only non-linear component, but the hardware implementation is complicated. For resource-constrained IoT devices, implementing algorithms with an SPN structure requires significant overhead. In contrast, algorithms with a Feistel structure use the same components for both encryption and decryption, thus consuming fewer resources. These algorithms are more suitable for resource-constrained environments. Cryptographic algorithms based on ARX exhibit non-linearity, diffusion and confusion through a combination of Addition or AND, Rotation and XOR. The simple structure of ARX makes it ideal for lightweight block ciphers. However, due to the particular features of ARX (Addition or AND, Rotation, XOR) operations, its round function can only be based on a Feistel structure or a generalized Feistel structure, otherwise, the decryption process cannot be completed correctly [12].

Traditional Feistel structures divide a plaintext source into two halves and apply a round function to half of the state before adding the result to the other half. The algorithm based on the Feistel structure is symmetrical and easily implemented in hardware and software. Without needing a specially designed decryption function, this kind of algorithm saves half of the hardware implementation design. However, generally, traditional Feistel structures do not have a good diffusion effect because they only process half the data in each round. Consequently, more rounds are required to ensure security.

A generalized Feistel structure [13] divides the input into k ($k \geq 2$) sub-blocks and potentially applies different F functions to each sub-block. Compared with a traditional Feistel structure, the diffusion effect of the generalized Feistel structure is slightly improved, but still cannot reach an ideal diffusion effect. Therefore, a new logical combination of a generalized Feistel structure and an ARX operation is proposed to enhance the diffusion speed, reduce iteration rounds and improve hardware performance. The main contributions of this paper are as follows:

1. To improve the diffusion effect of the traditional Feistel structure, a new variant of the generalized Feistel structure GFRX is proposed. In GFRX, two ARX structures

with different linear components are used to deal with all branches of the generalized Feistel structure to enhance diffusion and confusion.

2. To reduce the cost of hardware implementation, the decryption process of the proposed GFRX is similar to its encryption process, so significant additional resources are not required. Meanwhile, the encryption structure is reused in the key extension to minimize the additional resource consumption.
3. To improve the flexibility and efficiency of hardware implementation, different levels of serialization implementation are proposed to ensure efficiency under different hardware and throughput requirements.

The remainder of this paper is organized as follows: In Section 2, related studies are reviewed. In Section 3, we describe the specific details of the GFRX algorithm. In Section 4, the security of the GFRX algorithm is analyzed. In Section 5, we present performance results for the GFRX algorithm. Finally, we conclude the paper in Section 6.

2. Related Work

Lightweight block ciphers with a Feistel structure that have been proposed in recent years show excellent performance. SLIM [14] adopts a traditional Feistel structure and uses 4×4 S-boxes as non-linear components in the round function, which results in high security and excellent hardware performance. SAND [15] uses a combination of traditional Feistel and ARX structures, with a novel design method. The core idea of SAND is to limit the AND-RX operations to half bytes. Therefore, SAND enables equivalent representation based on a 4×8 synthetic S-box (SSb) in the security analysis, which greatly reduces the complexity of security analysis and results in strong software performance. However, the traditional Feistel structure makes the diffusion effect worse. Therefore, to address the diffusion effect, SLIM32/64 encrypts 32 rounds, while SAND64/128 encrypts 54 rounds. The high number of iterative rounds will inevitably lead to huge energy consumption.

Two methods have been introduced to improve the diffusion effect of the traditional Feistel structure. One uses complex round functions, such as in MIBS [16], μ^2 [17] and LiCi [18]. MIBS uses a complete SPN structure in the round function. Although strong diffusion is obtained, it offers advantages in terms of hardware consumption. The round function of μ^2 uses a four-round ultra-lightweight cipher for higher security, but its hardware implementation is more complicated. LiCi uses the SPN structure directly on the branch of the Feistel structure. Therefore, its encrypted branch looks like an independent SPN structure. However, this design method destroys the consistency of encryption and decryption for the Feistel structure, which requires an independent decryption module. Another approach uses the generalized Feistel structure constructed from the structure itself to improve the diffusion effect. Piccolo [19] uses a four-branch generalized Feistel structure with a more complex arrangement for the diffusion layers. TWINE [20] is a generalized Feistel structure encryption algorithm with 16 branches. The plaintext is divided into more sub-blocks in the encryption and the round function is used for the key extension. QTL [21] uses a four-branch generalized Feistel structure to process all branches in one round of encryption, which produces a very fast diffusion rate. However, to reduce the hardware cost, the QTL does not have a key extension function. This means that the same key is used in multiple iterations of the QTL, which makes the algorithm less resistant to standard statistical attacks. Shadow [12] is constructed of a combination of generalized Feistel and ARX structures, which enhances the diffusion of traditional Feistel structures. However, the key extension of the Shadow requires a lot of resources, which means that it offers no significant advantage in terms of hardware implementation. Therefore, hardware consumption should be reduced while improving the diffusion effect, especially in processing key extensions, which is addressed in this paper.

3. Specification of the GFRX

The proposed GFRX is based on a generalized Feistel structure with four branches. It supports seven combinations of plaintext length ranging from 64 to 128 bits and key

lengths ranging from 96 to 256 bits. The available block size, key size and corresponding rounds are shown in Table 1.

Table 1. Rounds of different block size.

Block Size	Key Size				
	96	128	144	192	256
64	26	27	-	-	-
96	28	-	29	-	-
128	-	32	-	33	34

- : Rounds do not exist, no such combination of block size and key size.

3.1. Encryption

In this algorithm, the ARX structure is used as a round function of a generalized Feistel structure, which makes the encryption more efficient in hardware, and ensures its diffusion and confusion. The GFRX algorithm includes four main operations: AND, ADD, Rotation and XOR. The structure of the proposed GFRX algorithm is shown in Figure 2.

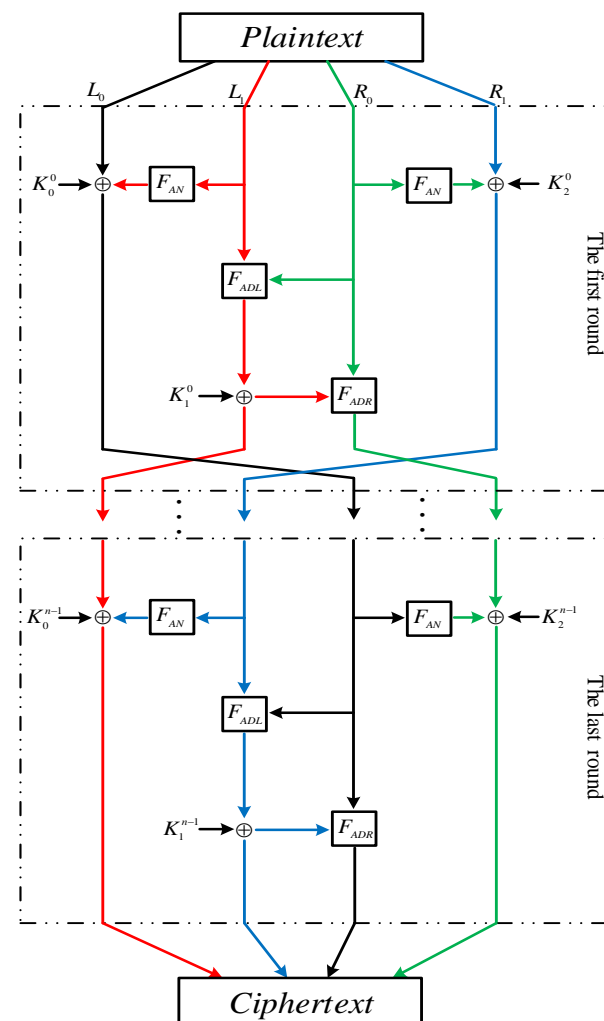


Figure 2. The overall structure of the GFRX algorithm.

In the encryption process, the plaintext block is divided into four sub-blocks of the same size. With the help of the primary key, the ciphertext is generated through multiple iterations. The GFRX algorithm uses two different round functions, which are based on

the ARX structure with different non-linear components. According to the difference between the non-linear components AND and ADD in the ARX structure, these two round functions are represented as F_{AN} and F_{AD} , respectively. Moreover, F_{AD} is divided into F_{ADL} and F_{ADR} . In a round of encryption, half of the branches are processed by the F_{AN} function and the rest are processed by the F_{AD} function. Then, a branch replacement operation is required. Branch replacement means that the branches processed by F_{AN} in the current round will be processed by F_{AD} in the next round instead. This method ensures that the ARX structure with two different non-linear components is fully used in the whole encryption process, which greatly improves the security and diffusion speed of the algorithm. The GFRX algorithm is shown as follows:

$$\begin{aligned} L_0^{i+1} &= F_{ADL}(L_1^i, R_0^i) \oplus K_1^i, \\ L_1^{i+1} &= F_{AN}(R_0^i) \oplus R_1^i \oplus K_2^i, \\ R_0^{i+1} &= F_{AN}(L_1^i) \oplus L_0^i \oplus K_0^i, \\ R_1^{i+1} &= F_{ADR}(L_0^{i+1}, R_0^i). \end{aligned} \quad (1)$$

The F_{AN} function is shown in Figure 3, referring to Equation (2).

$$F_{AN}(X) = (X \lll a) \& (X \lll b) \oplus (X \lll c). \quad (2)$$

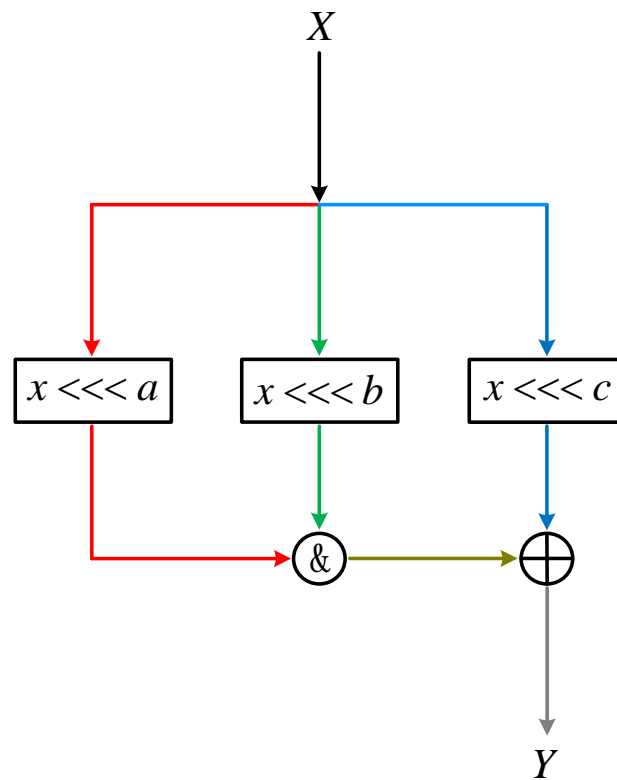


Figure 3. The flow of F_{AN} function.

As shown in Figure 4, function F_{ADL} is on the left and F_{ADR} is on the right. Functions F_{ADL} and F_{ADR} are separated from function F_{AD} and their relationship is represented by Equation (3):

$$\begin{aligned} F_{ADL}(X, Y) &= (X \ggg d) \boxplus Y, \\ F_{ADR}(Z, Y) &= (Y \lll e) \oplus Z, \\ F_{AD}(X, Y) &= F_{ADR}(F_{ADL}(X, Y), Y), \end{aligned} \quad (3)$$

where a, b, c, d, e in the above equation are called shift parameters; \boxplus indicates addition modulo 2^n . The pseudo-code of the GFRX encryption process is shown in Algorithm 1.

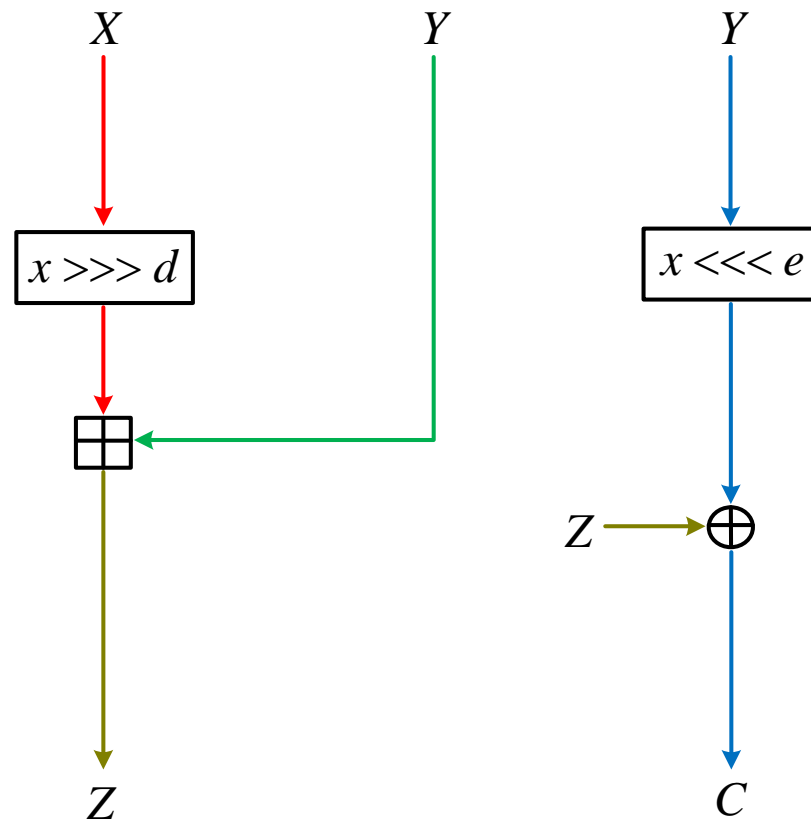


Figure 4. The flow of F_{ADL} and F_{ADR} .

Algorithm 1 GFRX Encryption.

Input: Plaintext, key

Output: Ciphertext

- 1: $(L_0, L_1, R_0, R_1) \leftarrow \text{Plaintext}$
 - 2: **for** $r=1$ to RN **do**
 - 3: $state0 = (L_1(\lll 1) \& L_1(\lll 8)) \oplus L_0 \oplus L_1(\lll 2) \oplus key_0^r$
 - 4: $state1 = (L_1(\ggg 8) + R_0) \oplus key_1^r$
 - 5: $state2 = R_0(\lll 3) \oplus state1$
 - 6: $state3 = (R_0(\lll 1) \& R_0(\lll 8)) \oplus R_1 \oplus R_0(\lll 2) \oplus key_2^r$
 - 7: $L'_0 = state1$
 - 8: $L'_1 = state3$
 - 9: $R'_0 = state0$
 - 10: $R'_1 = state2$
 - 11: **return** Ciphertext $\leftarrow (L'_0, L'_1, R'_0, R'_1)$
-

In the final step of each encryption, a branch replacement operation is required. The process of branch replacement is shown in Figure 5. Before the branch replacement, the X is $x_0 \| x_1 \| x_2 \| x_3$; after the replacement it is $x_1 \| x_3 \| x_0 \| x_2$.

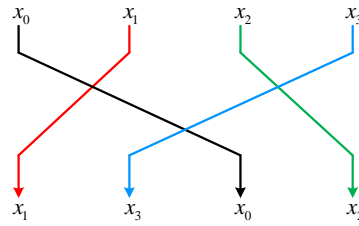


Figure 5. The process of branch replacement.

3.2. Key Extension

To reduce the hardware consumption of the key extension as much as possible, the round function in GFRX is reused to generate the required key. Specifically, F_{AN} is used to generate the K_1^i in function F_{AD} , while F_{AD} is used to generate key K_0^i and K_2^i required for function F_{AN} . Each round of GFRX requires three subkeys. The length of the subkey is the same as the GFRX branch n . The subkey required for the first round is obtained directly from the primary key. The subkeys required for the remaining rounds are generated by key extension. During the key extension, the input key K is first cyclically shifted left by $n * 3/2$ bits and subsequently divided into m parts of length n . Then, the m parts are divided into two blocks. Consequently, the key K can be expressed as $(k_{l_{m/2-1}} \parallel \dots \parallel k_{l_0}) \parallel (k_{r_{m/2-1}} \parallel \dots \parallel k_{r_0})$. Finally, the corresponding operation is used to process K to generate the three subkeys required in the round. The details of the key extension can be expressed by Equation (4):

$$\begin{aligned} k_{l_1} &= ((k_{l_1} \ggg d) \boxplus k_{l_0}) \oplus c \oplus z_i, \\ k_{l_0} &= (k_{l_0} \lll e) \oplus k_{l_1}, \\ k_{r_0} &= F_{AN}(k_{r_1}) \oplus k_{r_0} \oplus c \oplus z_i, \end{aligned} \quad (4)$$

where c is a constant, i.e., $c = 2^n - 4$; z_i is the i th bit of the m-sequence z . The m-sequence z with period 31 can be generated by the primitive polynomial $x^5 + x^2 + 1$ with the initial state $(1, 1, 1, 1, 1)$ of LFSR. When the round number is larger than 31, the sequence repeats itself. In contrast to the encryption when the round function is used in the key extension, constant c and m-sequence z_i are used to replace part of the key that was originally introduced to the round function.

Using this design method to generate round keys has two advantages. One is that the possibility of generating weak keys is avoided by using a complex round function. The other is that the hardware consumption is effectively reduced during the key extension so the round function does not need to rebuild new components. However, this method has its drawbacks. In each round of encryption, the round key is generated by the round function at first, then transferred to the round function to complete the whole process. This serial process inevitably reduces the throughput of the encryption algorithm. Lightweight encryption algorithms are usually used in harsh hardware environments with low throughput requirements. Therefore, it is feasible to sacrifice some throughput to reduce hardware consumption.

3.3. Decryption

The decryption process of the GFRX algorithm is similar to its encryption. Assuming that the algorithm is partially symmetric in the F_{AN} function with a reusable component, this means that the decryption can be performed by directly reusing the encryption structure of the F_{AN} function when decrypting to the part. Therefore, the decryption process of the GFRX algorithm only needs to be equipped with inverse functions of the F_{ADL} and F_{ADR} ,

which does not need many additional resources. The corresponding inverse functions F_{ADL}^- and F_{ADR}^- are shown in Equation (5):

$$\begin{aligned} F_{ADL}^-(X, Y) &= (X \boxminus Y) \lll d, \\ F_{ADR}^-(X, Y) &= (X \oplus Y) \ggg e, \end{aligned} \quad (5)$$

where \boxminus indicates the reduction modulo 2^n . One round of the GFRX decryption process is shown in Figure 6. It should be noted that the order of using the round keys in the decryption process is reversed.

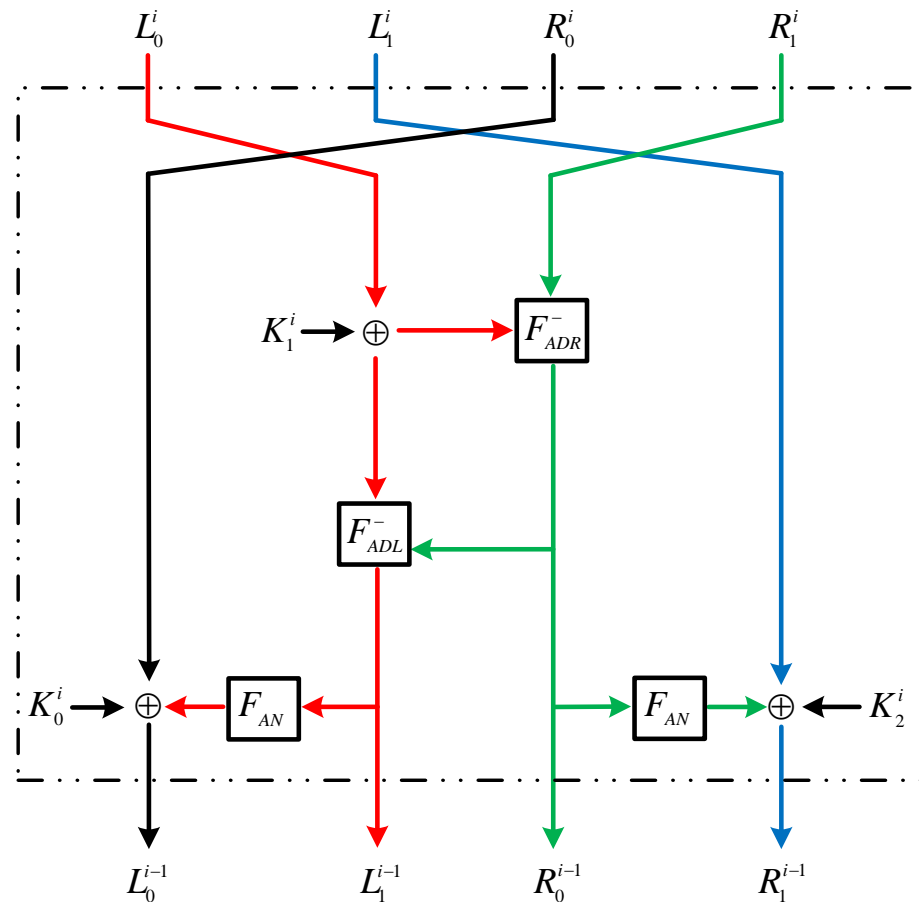


Figure 6. The i -th round decryption process of the GFRX algorithm

4. Security Analysis

In the encryption of the GFRX algorithm, branches affected by F_{AN} in the current round are processed by F_{AD} in the next round through branch replacement. The GFRX algorithm can be classified into a CFB [22] encryption mode. In each round, the encryption result of one branch is XOR with the other used to obtain the ciphertext. The ciphertext obtained in the previous round will be encrypted again in the next round, where the result will be XOR with the plaintext. The two ARX structures with different non-linear components can be used by all branches in the whole encryption process through branch replacement. The CFB mode makes each ciphertext block depend on all previous plaintext blocks. Even if the data of a ciphertext block is cracked, it is difficult to crack other data blocks by the same method, which greatly improves the algorithm's security.

Differential analysis [23] and linear analysis [24] of cryptography algorithms are effective attacks against iterative block ciphers. Any block cipher should be tested by both differential analysis and linear analysis.

4.1. Overall Structure Analysis

The proposed GFRX algorithm is based on a generalized Feistel structure, which is widely used in block ciphers and has good structural security [25]. In terms of encryption, GFRX uses the CFB encryption mode slightly differently from traditional CFB. The difference is that in the CFB mode of GFRX, the encryption function of each round is different. In terms of core components, the non-linear components of GFRX are implemented through the ARX structure, which also appears in SIMON and SPECK [26]. In recent years, a large number of studies has demonstrated that the components used by the SIMON and SPECK algorithms are sufficiently secure [27–29]. Therefore, the GFRX algorithm with a generalized Feistel and ARX structure can ensure adequate security.

4.2. Differential and Linear Analysis

The GFRX algorithm is divided into left and right, represented by $GFRX_L$ and $GFRX_R$, respectively. Since the left and right parts of GFRX use non-linear components which are similar to those of SIMON, whose security has mainly been investigated through differential and linear analysis, differential and linear analysis of the GFRX algorithm can utilize the analysis results for the existing SIMON algorithm.

In addition, in the encryption process of GFRX64/128, the branch replacement causes the left and right parts to interact with each other, making it more resistant to differential and linear analysis than the two independent GFRX32/64.

Taking GFRX64/128 as an example, GFRX64/128 can be divided into mutually independent $GFRX_L32/64$ and $GFRX_R32/64$. The GFRX64/128, which is resistant to differential and linear analysis, can be evaluated with the two independent GFRX32/64. Compared to SIMON32/64, GFRX32/64 has higher security with no unprocessed branches in each round of encryption. The evaluation of differential and linear security depends on the availability of efficient differential and linear trails. The authors of [30] proposed a technique for automatically searching the differential trails in ARX ciphers, called threshold search. The threshold search screens the differentials in the DTT (differential distribution table) and keeps those differentials whose probability is higher than a fixed probability threshold to form the partial DDT (pDDT). The authors of [31] improved the threshold search algorithm and divided the pDDT more finely to form a primary pDDT and a secondary pDDT. Subsequently, [31] used the improved threshold search algorithm to obtain the 13-round effective differential trails of SIMON32/64. On this basis, it was extended for six rounds and 19 rounds of attacks were carried out on SIMON32/64 using differential analysis with a time complexity of 2^{34} and a data complexity of 2^{31} . Most of the subsequent differential analyses for SIMON32/64 have been extended from the known 13-round differential trails. By reducing the complexity, more rounds can be used to attack SIMON32/64. In other words, the continuous differential trails for GFRX32/64 do not exceed 13 rounds, and the effective differential attacks do not exceed 19 rounds, which means that a complete GFRX64/128 is secure against differential analysis. Similarly, the effective linear trails of 11 rounds for SIMON32/64 are presented in [29]. Then, in [32], the result is extended by two rounds and uses linear analysis with the data complexity of 2^{32} to attack SIMON32/64 for 13 rounds. Thus, the continuous linear trails for GFRX32/64 do not exceed 11 rounds and the effective linear attacks do not exceed 13 rounds. The entire round of GFRX64/128 can withstand linear analysis.

5. Performance Evaluation

5.1. Avalanche Effect

The avalanche effect is an ideal property in cryptography. A good encryption algorithm must satisfy the avalanche effect. In a cryptographic algorithm, the avalanche effect can be defined as: when the plaintext changes by 1 bit, the ciphertext changes by about half. The average avalanche range is equal to the average value of the avalanche range on all bits. If an encryption algorithm satisfies the avalanche effect in a few rounds, it is considered

that its diffusion speed is fast. The avalanche effect test procedure for the GFRX algorithm is shown as follows:

1. A set of keys K is fixed randomly, then a set of plaintexts P is selected randomly.
2. After encryption, the initial ciphertext C is obtained.
3. The first bit of the plaintext is reversed and the rest of the bits remain unchanged; then the algorithm is input together with the key.
4. After encryption, the ciphertext C_0 is obtained.
5. The Hamming weight of $C \oplus C_0$ represents the avalanche degree of the first bit in the plaintext.
6. Repeat the above steps to find the avalanche degree on all bits of P .
7. Repeat the preceding steps 1000 times. Each time, P is random and K is the initial fixed key.

After the above steps, 1000 sets of avalanche data are obtained. Then, the average avalanche degree of each bit in the GFRX algorithm is accumulated with the average value. The standard deviation is calculated as the average uncertainty of each bit in the GFRX algorithm. Taking GFRX64/128 as an example, its average effect is shown in Figure 7. The results of using the same method to test the avalanche effects of SIMON and SPECK are shown in Figure 8 and Figure 9, respectively. From the figures, the average avalanche degree for each bit of the above algorithms and the fluctuation range of data in the whole avalanche process can be seen.

In the test of the avalanche effect of the above ciphers, error bars are used to indicate the uncertainty of the measured data. The error bar is a line segment drawn along the direction that indicates the size of the measured value with the measured arithmetic mean as the midpoint. Half of the line segment length is equal to the uncertainty. The uncertainty reflects the degree of fluctuation in the diffusion process. The smaller the uncertainty, the more stable the algorithm is in the diffusion process. Table 2 shows the specific performance of the SIMON, SPECK and GFRX algorithms on the avalanche effect. In general, the closer the average avalanche of an algorithm is to half the length of the plaintext, the better its diffusion performance is. Correspondingly, the less uncertain the algorithm, the more stable the diffusion.

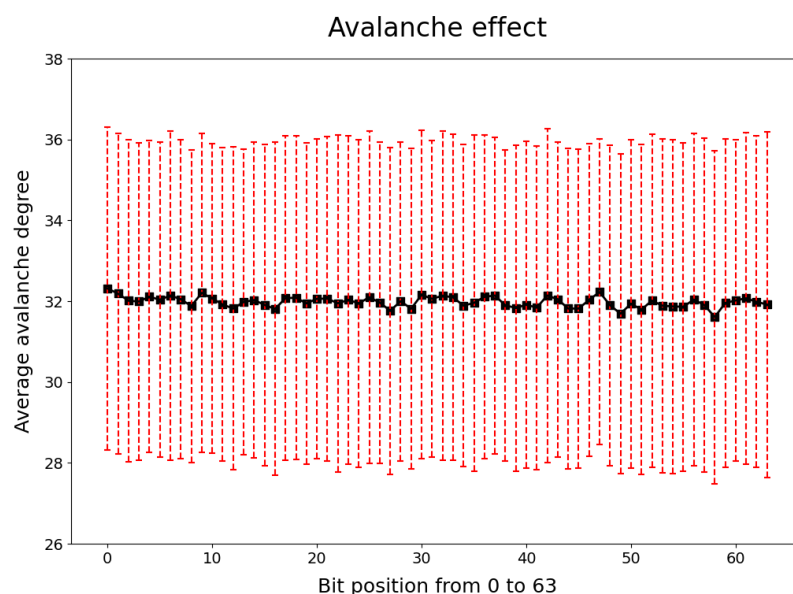


Figure 7. The average avalanche degree of GFRX64/128.

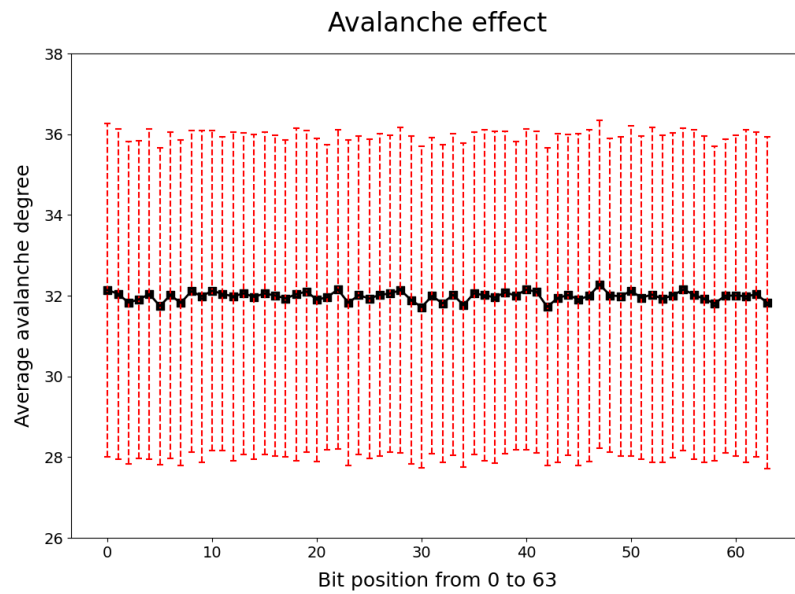


Figure 8. The average avalanche degree of SIMON64/128.

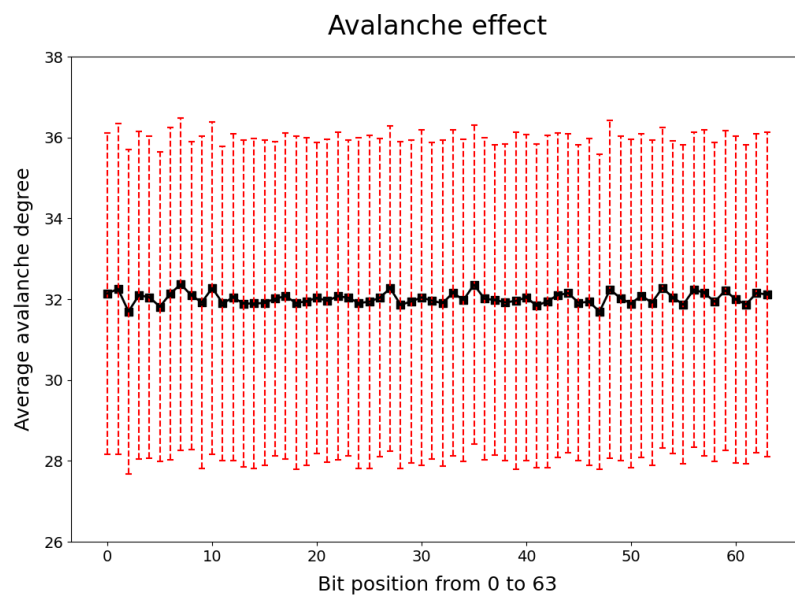


Figure 9. The average avalanche degree of SPECK64/128.

Table 2. Comparison of avalanche effect among structural algorithms.

Algorithm	Block Size	Key Size	Average Avalanche	Average Uncertainty	Avalanche Threshold
SIMON	64	128	31.970015	3.996816	12
SPECK	64	128	32.013781	4.000526	8
GFRX	64	128	32.005156	4.010121	6

It can be seen from Table 2 that the average avalanche degree of SIMON is less than 32 and the corresponding uncertainty is less than four. Meanwhile, the average avalanche

degree of SPECK is greater than 32 and the corresponding uncertainty is greater than four. The average avalanche degree of the GFRX algorithm is closest to 32 with the uncertainty slightly higher than four. The last column in Table 2 reflects the number of critical rounds to achieve the avalanche effect, which means that the lower the number of critical rounds, the faster the data will diffuse during the encryption. The minimum number of rounds for GFRX to achieve the avalanche effect is six. Thus, the GFRX algorithm exhibits a good avalanche effect.

5.2. Hardware Implementation

The GFRX algorithm uses a mixture of a generalized Feistel structure and an ARX structure. Compared with lightweight block ciphers using S-box, this algorithm improves the limitations of S-box in hardware implementation. It enables flexible serialization and achieves full serialization. The GFRX algorithm can be implemented with different serialization levels depending on the platforms' hardware requirements and different throughput requirements. Thus, this algorithm ensures efficient hardware implementation on different platforms. The serial architecture of GFRX used in the encryption module only is shown in Figure 10. In the figure, P_1 , P_2 , P_3 and P_4 represent the four branches of the initial data to be encrypted. In encryption, the combination of branches is controlled by 4-to-1 multiplexers.

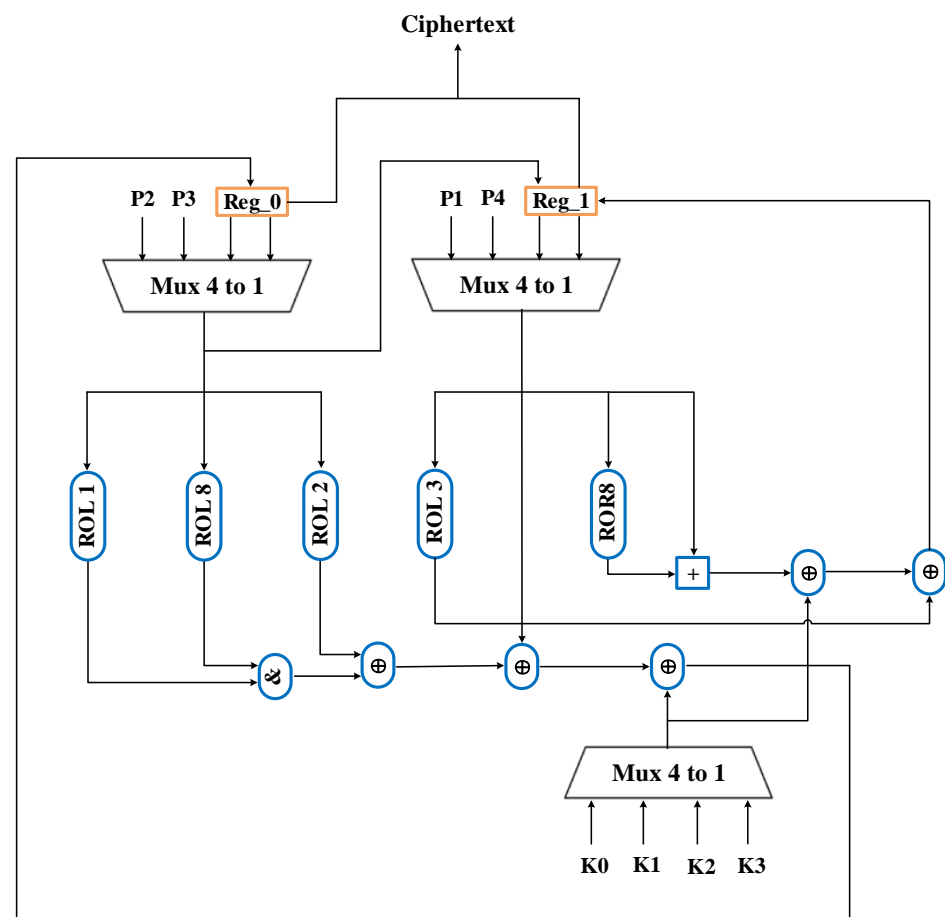


Figure 10. The serial architecture of the GFRX algorithm.

The performance of an encryption algorithm in hardware can be implemented by FPGA [33], and measured according to the required slice. Alternatively, it can be implemented by ASIC [34] and evaluated by the equivalent gate circuit GE. This section evaluates the specific GFRX64/128 in the above two ways. The results show that when encryption is implemented once per clock cycle on the FPGA Xilinx virtex-5 LX50T, the throughput is 246.15 Kbps with a hardware consumption of 4504 slices for 27 rounds of encryption

at a clock frequency of 100 kHz. Under the ASIC IBM130nm-8RF standard, the hardware consumption of a fully serialized GFRX algorithm is only 886.25 GE, while that of PRESENT64/128 [35] is 1886 GE. Therefore, it can be concluded that the GFRX algorithm is more suitable for use in resource-constrained environments.

5.2.1. FPGA Implementation

The GFRX64/128 algorithm is implemented by FPGA and the GFRX comprehensive download performance is analyzed by ISE14.7. Figure 11 shows a direct screenshot of the slices occupancy experimental data downloaded to the FPGA for the algorithm.

```

Design Summary:
Number of errors:      0
Number of warnings:    0
Slice Logic Utilization:
  Number of Slice Registers:      832 out of 28,800    2%
  Number used as Flip Flops:      832
  Number of Slice LUTs:          3,442 out of 28,800    11%
  Number used as logic:          3,442 out of 28,800    11%
  Number using O6 output only:    3,212
  Number using O5 and O6:         230
Slice Logic Distribution:
  Number of occupied Slices:      1,363 out of 7,200    18%
  Number of LUT Flip Flop pairs used: 3,515
  Number with an unused Flip Flop: 2,683 out of 3,515    76%
  Number with an unused LUT:       73 out of 3,515     2%
  Number of fully used LUT-FF pairs: 759 out of 3,515    21%
  Number of unique control sets:    102
  Number of slice register sites lost
    to control set restrictions:    304 out of 28,800    1%

```

Figure 11. Implementation of the GFRX algorithm on FPGA.

The simulation results show that the GFRX algorithm downloaded to the FPGA occupies 4504 slice units, 832 of which are occupied by slice registers and 3674 by LUTs. In FPGA implementation, the longer the plaintext and key length, the more slices are occupied for the same algorithm. Compared with PRESENT64/80, GFRX64/128 has a longer key length, requires less hardware resource consumption, and performs better on FPGA.

5.2.2. ASIC Implementation and Comprehensive Performance Evaluation

The unit GE is equal to the area required for a 2-input NAND gate with the lowest driving strength of the corresponding technology. The hardware consumption number of GE is equal to the total silicon area divided by the area occupied by the 2-input NAND gates. The GE uses the ARM standard unit library and the IBM 8RF(0.13 micron) ASIC manufacturing process. The GE consumption of some logic gates is shown in Table 3.

Table 3. Logical component resource consumption.

NOT	NAND	AND	OR	XOR	XNOR	4-1 MUX	D-Flip-Flop	1-Full-Adder
0.75	1.00	1.25	1.25	2.00	2.00	4.50	4.25	5.75

The GFRX algorithm is a lightweight block cipher that can be serialized flexibly. The degree of serialization has a significant impact on the cost and efficiency of the algorithm. The relationship between the degree of serialization and throughput can be calculated according to the following Equation:

$$throughput = \left(\frac{4L_b C}{4L_k + 3L_b R} \right) \delta. \quad (6)$$

where L_b and L_k in Equation (6) represent the length of the plaintext and key, respectively; C and R represent the frequency of the clock and encryption rounds, respectively; and δ indicates the serialization degree. When the serialization degree is 1, this implies full serialization. In the generalized Feistel structure, the degree of serialization does not exceed the branch length. After the branch length is exceeded, only non-serialized implementations can be performed. Fully serialized implementation has hardware consumption and the lowest throughput, while non-serialized implementation has the opposite. For a specific encryption algorithm, its plaintext length, key length, and encryption rounds are determined; the serialization degree is linearly related to the throughput, which increases as the serialization degree decreases.

The hardware consumption of the algorithm consists of two parts: GE_{other} and $GE_{combinational_circuit}$. When the plaintext block size and key size are determined, GE_{other} and $GE_{combinational_circuit}$ are fixed values. The relationship between the serialization degree and hardware consumption can be expressed by the following Equation:

$$GE_{cipher} = GE_{other} + GE_{combinational_circuit} \times \delta. \quad (7)$$

The performance of lightweight block cipher implementations can often be more accurately evaluated by combining a throughput and implementation area. The FOM [36] is usually used for evaluation. Its value is equal to the square of the throughput ratio over GE, which is shown in the following Equation:

$$FOM = throughput / area \text{ squared}. \quad (8)$$

Equation (9) can be deduced by combining Equations (6) and (7), i.e., the relationship between the FOM value and the degree of serialization.

$$FOM = \left(\frac{4L_b C}{4L_k + 3L_b R} \right) \delta / (GE_{cipher})^2. \quad (9)$$

When the length of the plaintext group and the length of the key group are fixed, many values in Equation (9) are constants, so it can be abbreviated to Equation (10).

$$FOM = k\delta / (a + b\delta)^2. \quad (10)$$

where $k\delta$ in Equation (9) indicates the throughput. Based on the analysis, the hardware consumption, throughput and FOM values according to different serialization levels can be obtained when the GFRX algorithm is implemented in ASIC hardware. Table 4 shows the hardware performance of GFRX64/128 at a clock frequency of 100 kHz.

Table 4. Performance comparison for different serialization degrees

Degree	Throughput (Kbps)	Area (GE)	FOM
1-bit	4.49	886.25	57.17
2-bit	8.98	926.25	104.67
4-bit	17.96	1006.25	177.38
8-bit	35.92	1166.25	264.09
16-bit	71.84	1486.25	325.22

The performances of some common lightweight block ciphers are compared and the results are listed in Table 5. The results in Table 4 and Table 5 show that the GFRX algorithm has low hardware consumption. In the case of 64-bit plaintext length and 128-bit key length, the maximum hardware consumption is only 1609GE, and the FOM value reaches the maximum 797.20, which exceeds that of some existing lightweight block ciphers.

Table 5. Performance comparison of common lightweight block cipher.

Algorithm	Throughput (Kbps)	Area (GE)	FOM
AES-128	12.40	3400	10.73
DES	44.40	2309	83.28
DESL	44.40	1848	130.00
Hight-128	188.25	3048	202.63
PRESENT-128	200.00	1886	562.27
GFRX-128	206.45	1609	797.20

6. Conclusions

In this paper, we propose a lightweight block cipher GFRX, combining a generalized Feistel structure and an ARX structure. The algorithm is based on a generalized Feistel structure with two different non-linear components of the ARX structure as round functions. In the round function, operations such as AND, ADD, Rotation, and XOR replace the hardware to effect the complex non-linear components. The flexible combination of generalized Feistel and ARX structures solves the problems of slow diffusion and confusion in traditional Feistel structures and offers great flexibility in hardware implementation. Compared with the current generalized Feistel structure algorithms, the GFRX algorithm has better diffusion and confusion effects, fewer iterations and higher hardware efficiency. The security analysis results for the GFRX algorithm show that the effective differential attacks do not exceed 19 rounds and the effective linear attacks do not exceed 13 rounds. Therefore, the GFRX algorithm is secure against differential and linear analysis.

However, there is no such thing as the best encryption algorithm for a specific application scenario, only the most appropriate one. The GFRX algorithm proposed in this paper can consume fewer resources with a sufficient security margin. Therefore, it has greater applicability in resource-constrained environments.

Author Contributions: Conceptualization, X.Z. and S.T.; Methodology, X.Z.; Software, S.T.; Validation, T.L., X.L. and C.W.; Formal analysis, X.Z. and S.T.; Investigation, X.Z.; Resources, X.L. and C.W.; Data curation, S.T.; Writing—original draft, X.Z.; Writing—review & editing, S.T. and T.L.; Supervision, X.L. and C.W.; Project administration, X.Z.; Funding acquisition, X.Z. and C.W. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Science Foundation of China under grant 61902156 and 62072217.

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

References

1. Ray, P.P.; Dash, D.; De, D. Edge computing for internet of things: A survey, e-healthcare case study and future direction. *J. Netw. Comput. Appl.* **2019**, *140*, 1–22. [\[CrossRef\]](#)
2. Goyal, P.; Sahoo, A.K.; Sharma, T.K.; Singh, P.K. Internet of things: Applications, security and privacy: A survey. *Mater. Today Proc.* **2021**, *34*, 752–759. [\[CrossRef\]](#)
3. Kakkar, L.; Gupta, D.; Tanwar, S.; Saxena, S.; Alsubhi, K.; An, D.; Noya, I.D.; Goyal, N. A secure and efficient signature scheme for iot in healthcare. *Cmc-Comput. Mater. Contin.* **2022**, *73*, 6151–6168. [\[CrossRef\]](#)
4. Rana, A.; Sharma, S.; Nisar, K.; Ibrahim, A.A.A.; Dhawan, S.; Chowdhry, B.; Hussain, S.; Goyal, N. The rise of blockchain internet of things (biot): Secured, device-to-device architecture and simulation scenarios. *Appl. Sci.* **2022**, *12*, 7694. [\[CrossRef\]](#)
5. Daemen, J.; Rijmen, V. Aes proposal: Rijndael. *AES Propos.* **1999**, *2*, 1–45.
6. Basu, S. International data encryption algorithm (idea)—a typical illustration. *J. Glob. Res. Comput. Sci.* **2011**, *2*, 116–118.
7. Al-Aali, Y.; Boussakta, S. Lightweight block ciphers for resource-constrained devices. In Proceedings of the 2020 12th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP), Porto, Portugal, 20–22 July 2020; pp. 1–6.
8. Bogdanov, A.; Knudsen, L.R.; Leander, G.; Paar, C.; Poschmann, A.; Robshaw, M.J.; Seurin, Y.; Vikkelsøe, C. *Present: An Ultra-Lightweight Block Cipher*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 450–466.

9. Banik, S.; Pandey, S.K.; Peyrin, T.; Sasaki, Y.; Sim, S.M.; Todo, Y. *Gift: A Small Present*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 321–345.
10. Liu, B.-T.; Li, L.; Wu, R.-X.; Xie, M.-M.; Li, Q.P. Loong: a family of involutational lightweight block cipher based on spn structure. *IEEE Access* **2019**, *7*, 136023–136035. [[CrossRef](#)]
11. Jha, P.; Zorkta, H.Y.; Allawi, D.; Al-Nakkar, M.R. Improved lightweight encryption algorithm (ILEA). In Proceedings of the 2020 International Conference for Emerging Technology (INCET), Belgaum, India, 5–7 June 2020; pp. 1–4.
12. Guo, Y.; Li, L.; Liu, B. Shadow: A lightweight block cipher for iot nodes. *IEEE Internet Things J.* **2021**, *8*, 13014–13023. [[CrossRef](#)]
13. Nyberg, K. Generalized feistel networks. In Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Singapore, 6–10 December 1996; pp. 91–104.
14. Aboushousha, B.; Ramadan, R.A.; Dwivedi, A.D.; El-Sayed, A.; Dessouky, M.M. Slim: A lightweight block cipher for internet of health things. *IEEE Access* **2020**, *8*, 203747–203757. [[CrossRef](#)]
15. Chen, S.; Fan, Y.; Sun, L.; Fu, Y.; Zhou, H.; Li, Y.; Wang, M.; Wang, W.; Guo, C. Sand: An and-rx feistel lightweight block cipher supporting s-box-based security evaluations. *Des. Codes Cryptogr.* **2022**, *90*, 155–198. [[CrossRef](#)]
16. Izadi, M.; Sadeghiyan, B.; Sadeghian, S.S.; Khanooki, H.A. Mibs: A new lightweight block cipher. In Proceedings of the International Conference on Cryptology and Network Security, Kanazawa, Japan, 12–14 December 2009; pp. 334–348.
17. Yeoh, W.Z.; Teh, J.S.; Sazali, M.I.S.B.M. $\mu 2$: A lightweight block cipher. In Proceedings of the Computational Science and Technology, Cagliari, Italy, 1–4 July 2020; pp. 281–290.
18. Patil, J.; Bansod, G.; Kant, K.S. Lici: A new ultra-lightweight block cipher. In Proceedings of the 2017 International Conference on Emerging Trends & Innovation in ICT (ICEI), Pune, India, 3–5 February 2017; pp. 40–45.
19. Shibutani, K.; Isobe, T.; Hiwatari, H.; Mitsuda, A.; Akishita, T.; Shirai, T. Piccolo: an ultra-lightweight blockcipher. In Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems, Nara, Japan, 28 September–1 October 2011; pp. 342–357.
20. Suzaki, T.; Minematsu, K.; Morioka, S.; Kobayashi, E. *Twine: A Lightweight Block Cipher for Multiple Platforms*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 339–354.
21. Lang, L.; Botao, L.; Hui, W. Qtl: A new ultra-lightweight block cipher. *Microprocess. Microsystems* **2016**, *45*, 45–55.
22. Ferguson, N.; Schneier, B.; Kohno, T. *Block Cipher Modes*; Wiley Online Library: Hoboken, NJ, USA, 2015; pp. 63–76.
23. Biham, E.; Shamir, A. *Differential Cryptanalysis of the Data Encryption Standard*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2012.
24. Matsui, M. Linear cryptanalysis of the data encryption standard. In Proceedings of the EUROCRYPT 1993, Lofthus, Norway, 23–27 May 1993; pp. 386–397.
25. Hoang, V.T.; Rogaway, P. *On Generalized Feistel Networks*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 613–630.
26. Beaulieu, R.; Shors, D.; Smith, J.; Treatman-Clark, S.; Weeks, B.; Wingers, L. The simon and speck lightweight block ciphers. In Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, 7–11 June 2015; pp. 1–6.
27. Abed, F.; List, E.; Lucks, S.; Wenzel, J. Cryptanalysis of the speck family of block ciphers. *Cryptology ePrint Archive* **2013**.
28. Alkhzaimi, H.A.; Lauridsen, M.M. Cryptanalysis of the simon family of block ciphers. *Cryptology ePrint Archive* **2013**.
29. Abed, F.; List, E.; Lucks, S.; Wenzel, J. Differential and linear cryptanalysis of reduced-round simon. *Cryptology ePrint Archive* **2013**.
30. Alex, B.; Vesselin, V. Automatic search for differential trails in arx ciphers. In Proceedings of the Cryptographers’ Track at the RSA Conference, San Francisco, CA, USA, 25–28 February 2014; pp. 227–250.
31. Biryukov, A.; Roy, A.; Velichkov, V. Differential analysis of block ciphers simon and speck. In Proceedings of the International Workshop on Fast Software Encryption, Istanbul, Turkey, 8–11 March 2015; pp. 546–570.
32. Alizadeh, J.; Alkhzaimi, H.A.; Aref, M.R.; Bagheri, N.; Gauravaram, P.; Kumar, A.; Lauridsen, M.M.; Sanadhya, S.K. Cryptanalysis of simon variants with connections. In Proceedings of the International Workshop on Radio Frequency Identification: Security and Privacy Issues, New York, NY, USA, 23–24 June 2015; pp. 90–107.
33. Nemati, A.; Feizi, S.; Ahmadi, A.; Makki, V.A.-d. A low-cost and flexible fpga implementation for speck block cipher. In Proceedings of the 2015 12th International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC), Rasht, Iran, 8–10 September 2015; pp. 42–47.
34. Mace, F.; Standaert, F.X.; Quisquater, J.J. Asic implementations of the block cipher sea for constrained applications. In Proceedings of the Third International Conference on RFID Security-RFIDSec, Amherst, MA, USA, 26–28 June 2007; Volume 2007, pp. 103–114.
35. Rolfes, C.; Poschmann, A.; Leander, G.; Paar, C. Ultra-lightweight implementations for smart devices—security for 1000 gate equivalents. In Proceedings of the International Conference on Smart Card Research and Advanced Applications, London, UK, 8–11 September 2008; pp. 89–103.
36. Manifavas, C.; Hatzivasilis, G.; Fysarakis, K.; Rantos, K. *Lightweight Cryptography for Embedded Systems—A Comparative Analysis*; Springer: Berlin/Heidelberg, Germany, 2013.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.