electronics

Article

A Huffman-Based Joint Compression and Encryption Scheme for Secure Data Storage Using Physical Unclonable Functions

Yong Liu [1], Bing Li [1,2,3,*], Yan Zhang [2] and Xia Zhao [1]

[1] School of Integrated Circuits, Southeast University, Nanjing 210096, China; liuyong2815@hotmail.com (Y.L.); eunicezhao@seu.edu.cn (X.Z.)
[2] School of Cyber Science and Engineering, Southeast University, Nanjing 211189, China; yanzhang930807@seu.edu.cn
[3] Shenzhen Research Institute, Southeast University, Shenzhen 518057, China
* Correspondence: bernie_seu@seu.edu.cn; Tel.: +86-1536-504-5432

Abstract: With the developments of Internet of Things (IoT) and cloud-computing technologies, cloud servers need storage of a huge volume of IoT data with high throughput and robust security. Joint Compression and Encryption (JCAE) scheme based on Huffman algorithm has been regarded as a promising technology to enhance the data storage method. Existing JCAE schemes still have the following limitations: (1) The keys in the JCAE would be cracked by physical and cloning attacks; (2) The rebuilding of Huffman tree reduces the operational efficiency; (3) The compression ratio should be further improved. In this paper, a Huffman-based JCAE scheme using Physical Unclonable Functions (PUFs) is proposed. It provides physically secure keys with PUFs, efficient Huffman tree mutation without rebuilding, and practical compression ratio by combining the Lempel-Ziv and Welch (LZW) algorithm. The performance of the instanced PUFs and the derived keys was evaluated. Moreover, our scheme was demonstrated in a file protection system with the average throughput of 473Mbps and the average compression ratio of 0.5586. Finally, the security analysis shows that our scheme resists physical and cloning attacks as well as several classic attacks, thus improving the security level of existing data protection methods.

Keywords: JCAE; PUFs; Huffman tree mutation; true random number

check for updates

Citation: Liu, Y.; Li, B.; Zhang, Y.; Zhao, X. A Huffman-Based Joint Compression and Encryption Scheme for Secure Data Storage Using Physical Unclonable Functions. *Electronics* **2021**, *10*, 1267. https://doi.org/10.3390/electronics10111267

Academic Editor: Filipe Araujo

Received: 26 April 2021
Accepted: 21 May 2021
Published: 25 May 2021

1. Introduction

The combination of Internet of Things (IoT) and Cloud-Computing [1,2] has emerged as a prospective platform, where IoT devices collect data from every corner of the real world and forward them to the cloud servers for the further analysis, and the cloud servers always have powerful resources to process them with high operational efficiency [3]. This revolutionary architecture greatly changes our lives by bringing various applications to the areas of Public Security, Intelligent Transportation, Smart Home, etc. However, the security and its efficiency against attacks of data storage in cloud servers have become two problems impeding the developments of this architecture. The amount of the data increases rapidly with the explosive growth of IoT devices, thereby making it an urgent requirement to keep a satisfactory operational efficiency for data storage [4]. Moreover, the security of data storage should be ensured, since this IoT data may involve the sensitive information, such as personal images, secret files, user passwords, and diagnostic data [5], etc. Therefore, it is vital of importance to develop an efficient and secure data storage scheme for cloud servers.

Joint Compression and Encryption (JCAE) has been regarded as an effective technique to supplement the existing data protection methods, such as Attribute-based Encryption (ABE) [6], Identity-based Encryption [7], and symmetry encryption [8] for cloud servers, as it not only provides a function of compression to achieve the good operational efficiency of the data storage, but also supports a function of encryption to further enhance the security level [9–11]. Moreover, the incorporation of this novel JCAE method would not affect the operational efficiency of data protection process. As a result, introducing JCAE scheme into data protection is considerable.

The construction method of JCAE has been proposed mainly by integrating the confusion and diffusion mechanism into the entropy coding compression algorithms, including Huffman codes [9–11] and arithmetic codes [12–14], which compress the data by utilizing a statistical model. To meet the requirements of encryption, a secret key will be chosen to modify the statistical model. For the huge computational costs of operations in both compression and encryption processes, Huffman coding algorithm and stream cipher algorithm are chosen for the low computational cost to construct the JCAE in which encryption does not affect the properties of the compressed data or the compression ratio. For example, the process based on multiple Huffman tables controls the exchange of the node branches of Huffman tree through the key and accomplishes the encryption and compression at the same time [9]. As a result, the output of the compression algorithm will become ciphers, as the initial statistical model has been changed. Only if the receiver obtains the secret key will he synchronize the modified statistical model to correctly decrypt the ciphers. In general, it is promising to incorporate this kind of key-controlled mechanism into the entropy coding compression algorithms to develop the JCAE scheme, thus improving the existing data protection systems for cloud servers.

Various studies [9–11,15,16] have put efforts on the developments of the JCAE based on the Huffman coding algorithm. Multiple Huffman Tables (MHT) Algorithm [9] uses the multiple Huffman coding tables when encoding for operational efficiency but proved vulnerable to known-plaintext and chosen-plaintext attacks in [17]. Chaotic Huffman Tree (CHT) Algorithm uses piecewise linear chaotic maps to obtain a key-stream to mutate and update the Huffman coding tree for security but is time-consuming. Swapped Huffman code Table (SHT) Algorithm updates the Huffman coding tree and encode plaintext segment by segment for security but is still not efficient in operation. Although the above research has been done, the following issues remain to be addressed:

1. The key-stream used in the JCAE is only pseudo-random number (PRN), which is not as secure as true random number (TRN).
2. The dynamic Huffman tree update operation is time-consuming and frequent, which makes the JCAE inefficient.
3. The compression ratio of the JCAE scheme could be further improved, which is defined as the ratio of the size of the compressed file to the original file in this paper.

PUFs can generate TRN and resist physical and clone attacks. The PUFs circuit widely existing in many devices, can provides lightweight encryption [18–20]. So, a Huffman-based JCAE for the secure data storage using PUFs is proposed for the first time to address the above issues. The contributions are as follows:

1. A physically secure key generator is incorporated into JCAE scheme by using the PUFs key is put forward. In this method, the Bistable Ring PUF (BR-PUF) circuit is instanced to generate TRN and further produce the PUFs key, which makes our JCAE scheme resist physical and cloning attacks.
2. A parallel mutation method based on the basic Huffman tree is proposed for operational efficiency by reducing the computational cost of the merging process and employing a parallel architecture. This method first generates the basic Huffman tree by the statistic model calculated from the input messages. Then, to update the Huffman coding table with the modified mutated Huffman tree technique, instead of rebuilding the new tree. Moreover, a parallel architecture is designed to execute the PUFs key generation and the basic Huffman tree construction at the same time. In general, the operational efficiency is improved.
3. A cascading encode structure is designed by inserting the LZW algorithm into our scheme. The introducing of LZW algorithm not only reduces the complexity of the input messages, but also helps our scheme maintain a satisfactory compression ratio, which makes it generic and practical to adapt to the existing data storage method.

4. The proposed JCAE scheme is implemented on the Field Programmable Gate Array (FPGA) Xilinx LX110T and then integrated into a proof-of concept file protection system. The performance evaluation shows that the proposed scheme is efficient in execution and has a practical compression ratio.
5. The security analysis is performed to show that our scheme can improve the original security level by resisting physical and cloning attacks, and several classic attacks.

The remaining part of this paper is organized as follows. In Section 2, the related JCAE schemes and BR-PUF are introduced. Section 3 described our proposed scheme in details. In Section 4, the performance of our implementation is evaluated. Then, the security analysis is carried out in Section 5. Finally, we conclude our work in Section 6.

2. Related Works

Recently, many cloud-related and IoT-based applications which concern the data security have been researched. We list the main contributions of the related works in Table 1.

Table 1. Summary of contributions of existing data security methods and security analysis.

Methods.	Year	Type	Specific Contributions
McIntosh et al. [21]	2021	Method	Their work defended malware attacks by proposing a situation-aware access control scheme, which supports to defer the access control decision and roll back the changes when necessary.
Kayes et al. [22]	2019	Method	They proposed a context-aware access control method to characterize the imprecise context for data from multiple cloud sources.
Sharma et al. [23]	2019	Analysis	They proposed a novel classification for cloud and IoT-based data models, discussed important cases studies, and detailed the emerging service and data analytics in the area of autonomous vehicles.
Xue et al [24]	2017	Method	They developed a collaborative access control scheme for outsourced data by using attribute-based encryption. Authorized multiple users can gain access permission by collaboration.
Hermassi et al [10]	2012	Method	They proposed a joint compression and encryption scheme by using the chaotically mutated Huffman trees.
Balduzzi et al [25]	2012	Analysis	They implemented an automated system to test and analyze the security of the Amazon public AMIs, and found the security problems, including unauthorized access, malware attacks, and leakage of sensitive information.
Somorovsky et al [26]	2011	Analysis	They performed a security analysis to the control interface of cloud service. The analysis shows that the control interfaces are at risk of signature wrapping and advanced XSS attacks.
Proposed scheme	2021	Method	Our scheme is the first to integrate the PUFs key into the Huffman-based compression to improve the security. Besides, a parallel mutation method as well as a cascading structure are developed to further improve the efficiency and support a good compression ratio.

The rest of this section will focus on the related works of the JCAE schemes based on Huffman and the PUFs used in the scheme.

2.1. Huffman Tree Mutation (HTM) Technique

In [9], a technique was proposed to create the Huffman coding tables with the compression ratio unchanged. They trained and obtained a basic Huffman tree. Then, hundreds of different tables could be derived by Huffman tree mutation. The basic Huffman tree had leaves and inner nodes. Each node had the left branch, normally labeled "0" and the right branch, normally labeled "1", which is called the label-pair. If the label-pairs were permuted, a new Huffman tree could be derived. This process is called the Huffman tree mutation as shown in Figure 1. The HTM technique provided the opportunity to decide how to permute every label-pair. To derive a new Huffman tree, a random bit-stream was generated, then the label-pairs of the basic Huffman tree were permuted if the corresponding bit was 0, otherwise kept unchanged. It is important to note that the Huffman tree mutation has no effect on the coding efficiency and had equal codeword lengths.

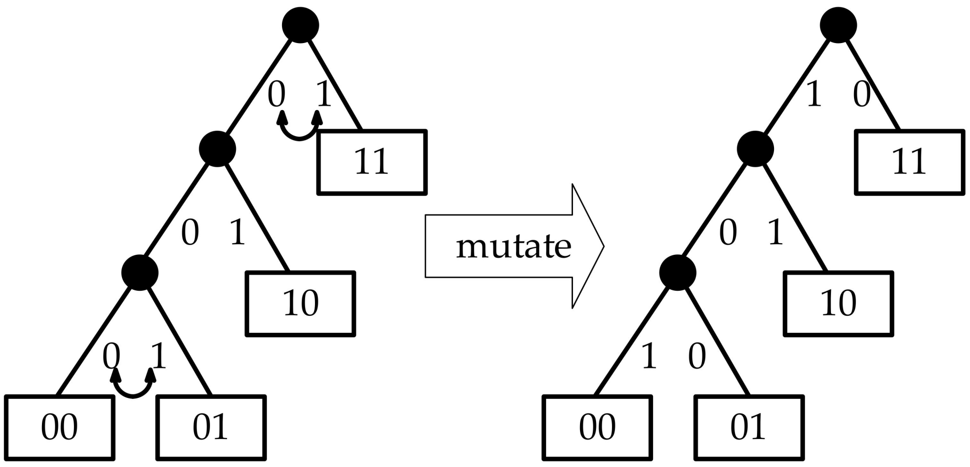

Figure 1. The Huffman tree mutation process.

2.2. Multiple Huffman Tables (MHT) Algorithm

MHT [9] was a scheme which combined compression and encryption with the multiple Huffman coding tables during the encoding process. The secret key consisted of m distinct Huffman coding tables and a key-stream ($k_0, k_1, \ldots k_{n-1}$) which value of k_i was selected from the set $\{0, 1, \ldots m-1\}$. The Huffman tables were randomly selected from the public pool of Huffman tables by key-stream to encode a symbol, so the adversary could not get the selected tables exactly.

Since the given symbols were encoded to the codewords with different lengths by different basic trees, the MHT algorithm affected the compression ratio. The serious problem is that "MHT are vulnerable to low complexity known- and/or chosen-plaintext attacks" [17] by the scheme proposed in [17].

2.3. Chaotic Huffman Tree (CHT) Algorithm

The CHT [10] used a chaotic map to generate the key-stream to update the Huffman coding tree. First a Huffman coding tee was generated "based on the probability distribution of the symbols in the message" [10]. The basic Huffman tree was shown in Figure 2a. This Huffman tree had 6 symbols, which means 6 "leaves" and 5 "inner-nodes" in the tree. The leaves were represented by "points" and the inner-nodes were represented by "squares" and they were numerated from (1) to (5). Permuted the label-pairs of selected inner-node according to the key-stream with HTM technique and updated the Huffman tree as shown in Figure 2b.

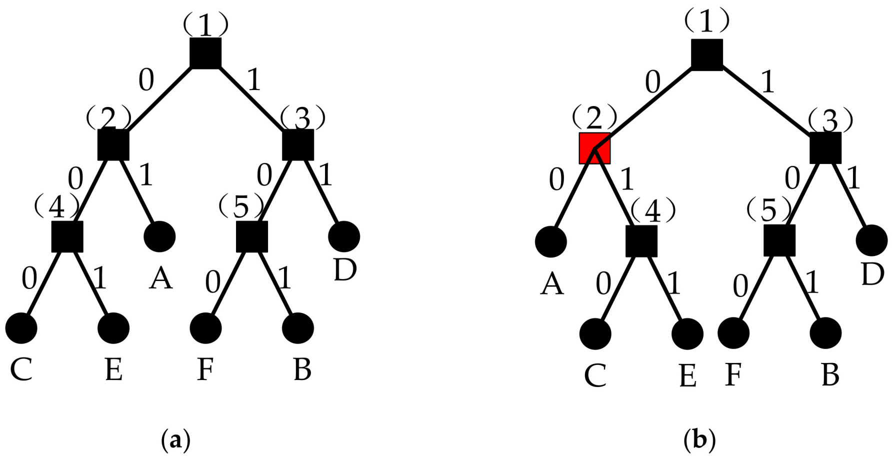

Figure 2. (**a**) Basic Huffman tree; (**b**) Mutated Huffman tree by changing the label-pair of inner-node (2).

The CHT must update the Huffman tree each time for each symbol. It would increase the computation cost. As described in [10], the additional average CPU instructions over the standard Huffman coding can be calculated by $cost = (2 + n + 5 + N)/8$ per bit, where n was the literation number and N was the number of symbols. For example, Calgary corpus is a collection of files, and used to compare the data compression algorithms. The test file "paper5" in the Calgary corpus got $n = 10$ and $N = 256$, and the number of the cost would be 34 CPU instructions per bit. So, CHT maintains efficient coding but not an execution efficient scheme.

2.4. Swapped Huffman Code Table (SHT) Algorithm

The SHT [11] used the key-stream to update the Huffman coding tree and encode plaintext segment by segment as shown in Figure 3.

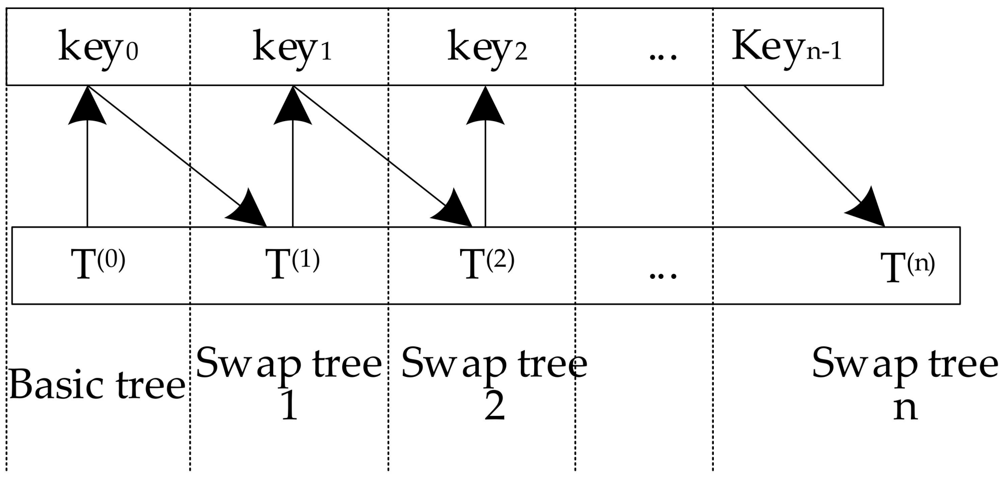

Figure 3. Swapped Huffman code Table.

First, the Huffman coding tree based on the statistical model of the plaintext symbols is generated. Then each time we got N symbols in the plaintext, the Huffman tree was mutated and updated using key-stream with HTM technique. The mutation would occur when every N symbol was obtained, which could reduce the computational-cost. However, every update operation will still cost extra memory-load, comparison, exchange, and memory-store operation. In other words, SHT still is not an efficient execution scheme.

2.5. Bistable Ring PUF (BR-PUF)

PUFs is a noise function of the physical entity, which implements the mapping from inputs (challenges) to outputs (responses). This mapping has randomness and uniqueness, which inevitably result from the uncontrollable differences of physical entity during the manufacturing process. PUFs can be divided into strong PUFs and weak PUFs according to the space of challenge-response pairs (CRPs). BR-PUF [27] structure is exactly one of the hopeful candidates for strong PUFs.

BR-PUF is a combination of SRAM PUFs [28] and APUF [29]. It not only has millions of excitation space like APUF, but also has a bistable ring structure like SRAM PUFs. BR-PUF has an n-stage series loop structure. Each stage structure contains two NOR gates, one 2-input multiplexer and one 2-input demultiplexer and requires 1 bit challenge, as shown in Figure 4.

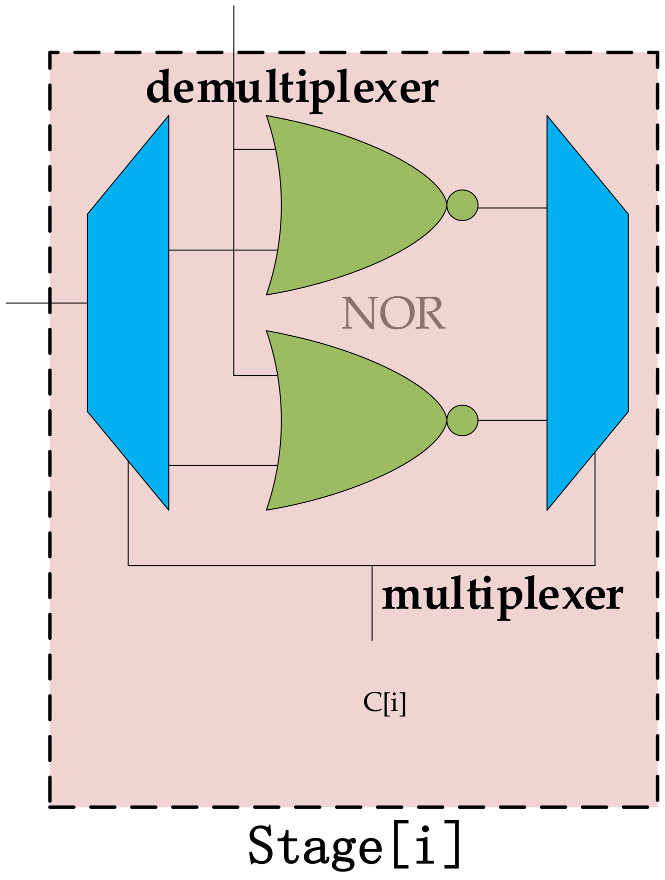

Figure 4. One stage structure of BR-PUF.

Figure 5 shows a bistable ring with 8-stage inverters [27]. After power on, each inverter in the ring will force the output from "0" to "1". However, due to the reaction of the differences of inverter threshold voltage, the inverter ring has two stable states, and the ring does not enter a stable state immediately after being energized. It will oscillate over a period before reaching a stable state which may be "10101010" or "01010101". When the ring enters a stable state, the output of one of the inverters is taken as the response of BR-PUF.

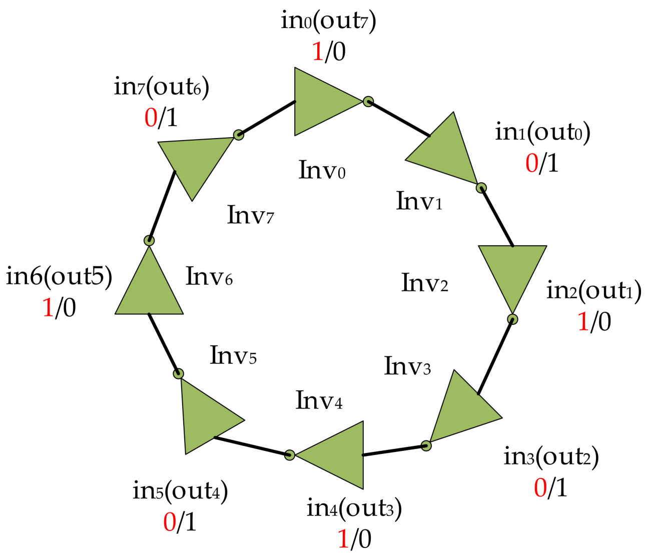

Figure 5. Illustration of bistable ring with 8-stage inverters.

3. Proposed Scheme

In this section, the overview of our scheme is first introduced to show the working flow of our scheme. Then, several modules formulating our proposed scheme are explained, respectively.

3.1. Overview

The block diagram of the compression-encryption progress is shown in Figure 6 and detailed as follows:

1. Key generation: A PUFs challenge $C \in \{0,1\}^{128}$ is generated and sent to the PUFs module to generate the correlated response $R \in \{0,1\}^{128}$. The response acting as a true random number is further input into the Huffman tree mutation module as key-stream.
2. Pre-encoding: The input message will first be pre-encoded by the LZW module to output the bit stream. Every 8 bits are constructed one symbol, and 32 symbols forms one block. In theory, one challenge can process one block. To improve the throughput, we reuse one challenge to encode M blocks. It will be security when M is less than 32 which suggested in [17].
3. Basic Huffman Tree (BHT) Construction: In this process, 32N symbols will be sent to the Statistics module. The Statistics module outputs the weight of every received symbol to construct a basic Huffman tree BHT with K inner-node.
4. Mutation: K bits from the key-stream generated in step 1 will be used to mutate BHT constructed in step 2, in order to obtain the corresponding mutation Huffman tree (M-HT). Each M-HT will compress/encrypt 1 block M times.
5. Cipher output: As is described in step 4, every challenge can generate 128 bits key-stream at once. Since each symbol is encoded with variable code length, the length of the output bit stream is not fixed. The challenge C will be attached to the beginning of the output bit streams. Finally, the challenge C and the correlated BHT will be recorded into a map C_BHT for the decompression-decryption.

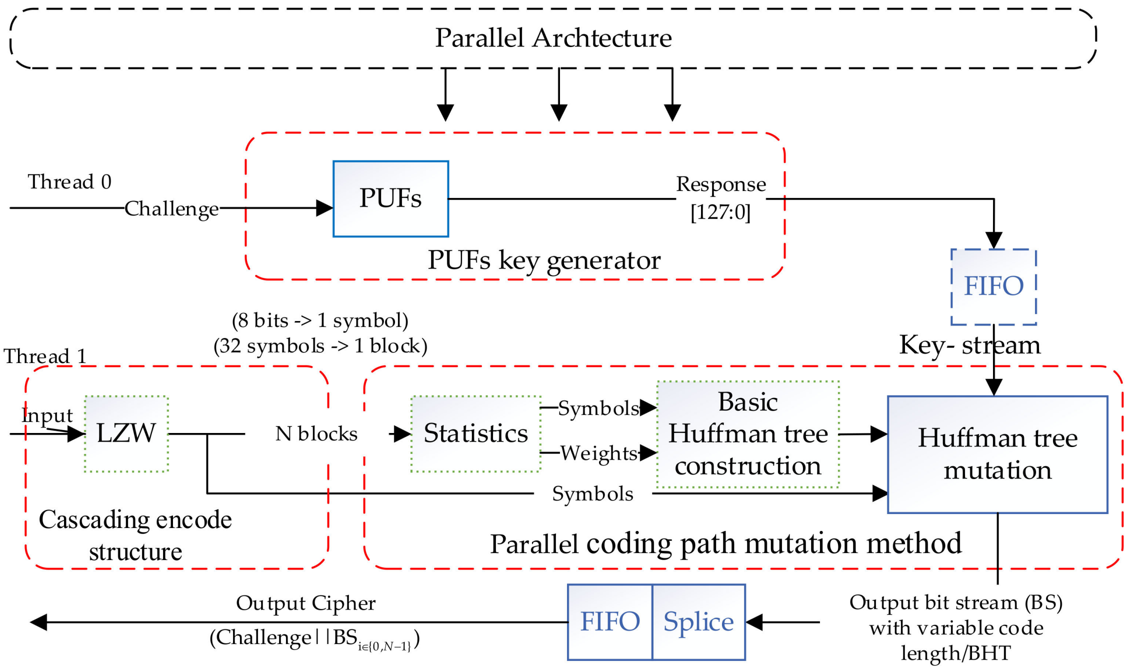

Figure 6. Block diagram of joint compress-encrypt process. The dotted green lines indicate that the procedures are only executed once; the solid blue lines indicate that the procedures are executed until the file transfer is complete.

It is noted that a parallel architecture is designed to execute our scheme in our hardware platform. As is shown in Figure 6, Thread 0 and Thread 1 will start at the same time and meet at the Huffman tree mutation module.

Then, the decompression-decryption progress is demonstrated in Figure 7 and we only introduce the main differences between these two processes:

1. Cipher separation: The cipher is composed of challenge C and the bit streams $BS_{i\in\{1,N-1\}}$. We separate the cipher and send them to different modules.
2. Key generation: This procedure remains unchanged according to the received challenge.
3. Basic Huffman Tree Construction: This process directly finds the BHT from the C_BHT map according to the received challenge, and then constructs the basic Huffman tree.
4. Mutation: This process remains unchanged.
5. Plaintext output: With the help of M-HT generated in step 4, the bit stream will be decompressed and decrypted into *N* blocks. In the final, these blocks are decoded by the LZW module to obtain the original input messages.

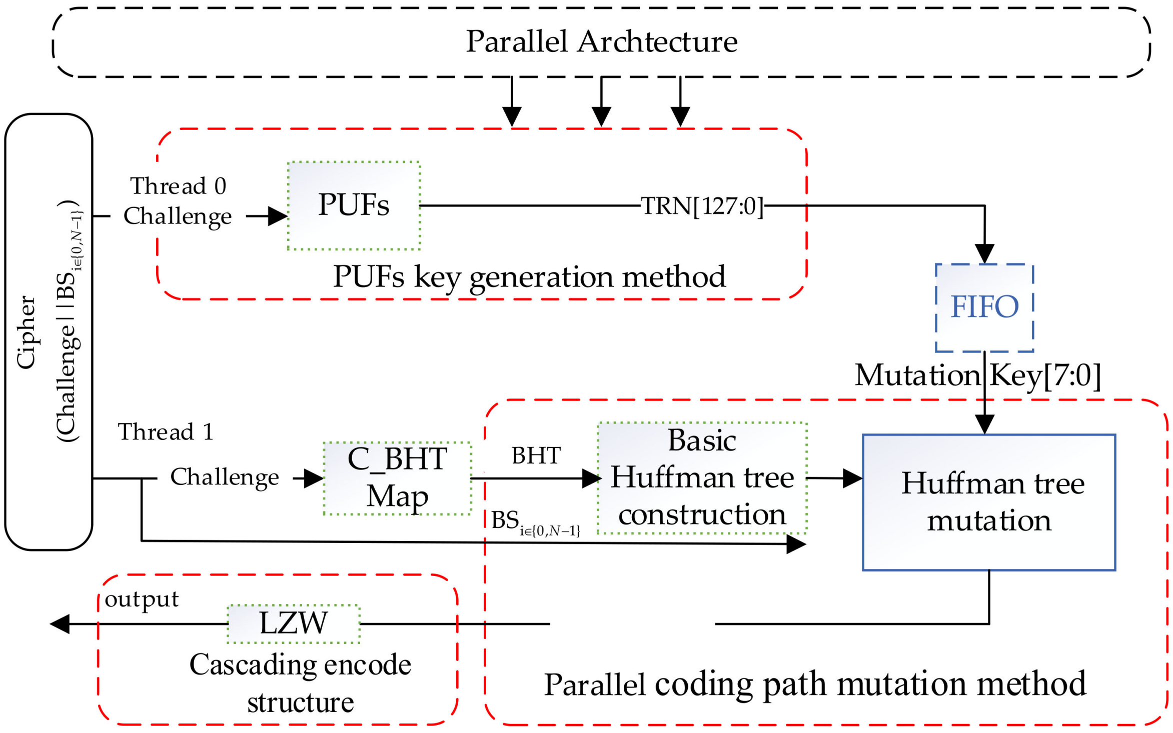
Parallel Archtecture
Cipher (Challenge | | $BS_{i\in\{0,N-1\}}$)
Thread 0
Challenge
PUFs
TRN[127:0]
PUFs key generation method
FIFO
Mutation Key[7:0]
Thread 1
Challenge
C_BHT Map
BHT
$BS_{i\in\{0,N-1\}}$
Basic Huffman tree construction
Huffman tree mutation
output
LZW
Cascading encode structure
Parallel coding path mutation method

Figure 7. Block diagram of joint decompress-decrypt process. The dotted green lines indicate that the procedures are only executed once; the solid blue lines indicate that the procedures are executed until the file transfer is complete.

After the overall description to our proposed JCAE scheme, we will introduce the n parts involved in our scheme, including PUFs key generation method, Statistic module, efficient mutation method, etc., respectively.

3.2. PUFs Module

We chose BR-PUF described in Section 2.5 to generate the key-stream. A 128-stage BR-PUF is implemented as described in Figure 8. Each stage has the structure shown in Figure 4 and shares a challenge C[*i*], which decides different signal path of the MUX and the DEMUX. Then the 128 stages NOR-gate form a ring exclusively with a 128-bit challenge. As the structure shows in Figure 8, 2^{128} different rings can be created.

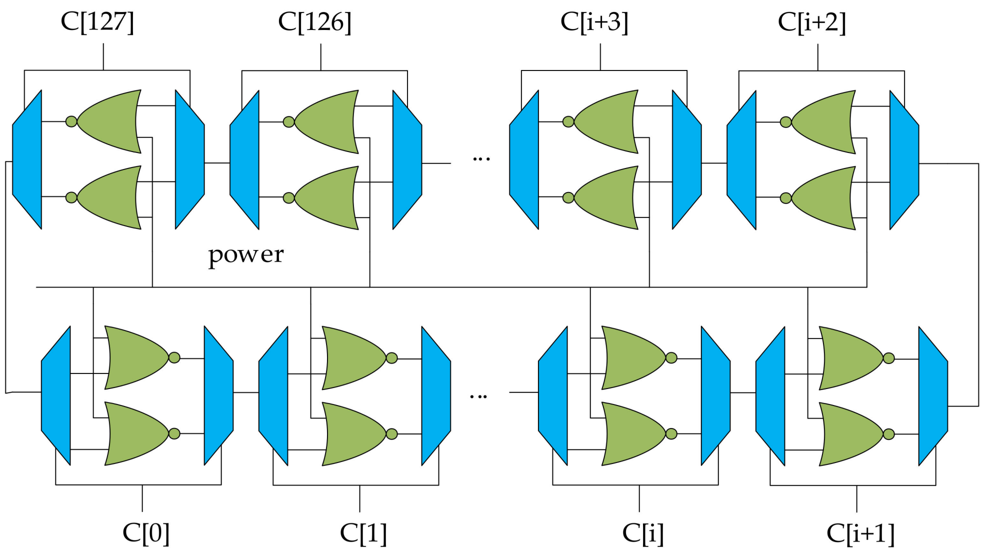

Figure 8. A 128-stage Bistable Ring PUF.

When powered on or reset, each stage of BR-PUF forces the output level from low to high. Due to the differences of each NOR-gate electrical characteristics, the NOR ring may have two stable states. After the NOR ring is energized, it will take a certain time to reach a steady state. When the NOR ring enters a steady state, the output is taken as the response of BR-PUFS. The response generation process of BR-PUF is shown in the following steps.

1. Set power = 1 to shut down the NOR ring.
2. Apply challenge signal Ci.
3. Wait for the NOR ring to the state of full "0".
4. Set power = 0 to energize the NOR ring.
5. Wait for n clock cycles to enter a stable state.
6. Read the output as a response.
7. Repeat step 1 to step 6.

3.3. LZW Modules

A dictionary compression algorithm called LZW is used to compress and decom-press data, which was proposed by Welch in 1984 on the basis of LZ78 (Lempel-Ziv).

The LZW module works with a dictionary, which encodes unique partial sequences into codewords. Ideally, l-bit length codewords can express 2*l*-bit length sequences [30]. LZW is used in conjunction with Huffman entropy coding algorithm in a variety of commercial compression software to further improve the compression ratio. When compressing (in Figure 6), it converts the sequences into codewords; when de-compressing (in Figure 7), the module reverses the conversion.

3.4. Statistic Modules

The statistics module is designed to analyze the character distribution (frequency) of the sequence to be compressed, in order to get a better (lower) compression ratio [15]. in Huffman coding algorithm. This module obtains the frequency distribution of symbols from the input stream. The message *M* will be separated and input into this module symbol by symbol. If the symbol appears the first time, it will be placed into a new register and the frequency counter will be assigned to 1. If the symbol is the same, the frequency counter should add 1. The frequency of the symbol is called weight normally. According to the frequency statistics, the characters with higher frequency will be assigned to the shorter depth of the Huffman tree nodes. Symbols and their weights will be delivered to BHT construction module as shown in Figure 6, and are further used to calculate the probability distribution of the symbols.

3.5. Basic Huffman Tree Construction Module

This module generates a basic Huffman tree based on the probability of the symbols calculated in the statistic module. BHT has various functions, including sorting, selection of minimum and second minimum, summation of minimum and second minimum, weight updating and binary tree generation. In this module, Sorting and selection operations are performed simultaneously in hardware implementation.

The sorting operation function is to sort the frequency of each symbol from small to large. In hardware implementation, the comparator array is constructed to compare every two numbers in pairs, and then the adder is used to sum the comparison result of the frequency of each symbol. For one symbol frequency, if it is less than the frequency of another symbol, the corresponding result is 0, otherwise it is 1. By specifying the sum of the symbol frequency and the other symbol frequency, the position of its frequency among all the frequencies can be determined.

The selection operation selects the symbol frequencies corresponding to the comparison results of 0 and 1 in the sorting operation, which represent the minimum frequency and the second minimum frequency. The selection operation also selects the symbols corresponding to the two frequencies. The hardware implementation builds multiple comparators, and then uses the multiplexer to accurately pick out the desired value from multiple frequency and symbols.

The sum operation of minimum value and sub-small value is to sum the minimum frequency and sub-small number selected by the selection operation to form a new node as the parent node of two leaf nodes, and the two leaf nodes are the left and right branch of the new node, respectively.

Binary tree and coding module: replace two leaf nodes with new nodes, reorder, select, sum, until a root node is generated, and finally generate a binary tree. The Huffman coding table is generated by reverse traversal from leaf node to root node.

3.6. Huffman Tree Mutation Module

After a basic Huffman tree is created, this module permutes the label-pairs of inner-node according to the key-stream to implement encryption. The Huffman coding algorithm is an entropy algorithm to assign shorter coding to frequently occurring symbols and longer coding to fewer occurring symbols. For a Huffman tree with N leaves, the average coding length of a symbol, I_{avg} is expressed as Equation (1):

$$I_{avg} = \sum_{1}^{N} l_i \cdot \log_2 p_i \tag{1}$$

where p_i is the probability of the ith symbol appearing in the source message. l_i is the coding length of the ith symbol got from the Huffman tree. As shown in Figure 9, the coding of symbol E changes from 001 to 010. It is obvious that all symbol coding only exchanges label values of branches at the corresponding inner-node, so the encoding length remains unchanged, and I_{avg}, remains unchanged. As mentioned above, encoding efficiency is expressed in terms of compression ratio. After the compression, the size of the data is equal to the total number of symbols multiplied by the average encoding length of the symbols. The total number of symbols, average coding length, and plaintext length remain unchanged, so the compression ratio is the same as that of the original Huffman algorithm, which achieves the best compression effect and ensures the same coding efficiency as the original Huffman algorithm.

The essence of inner-node mutation is to keep the tree structure unchanged, only exchange the label, and then change the symbol encoding in the coding table. This is a time-consuming process, requiring n-1 node mutation and n-code update operations. N-1 node Mutation is easy to be realized in hardware design. While the process of obtaining a new code table through subsequent reverse traverse takes a long time, with a complexity of o($nlog_2n$), which takes a long time and increases the calculation consumption each time.

For example, the experimental results of CHT show that the operational efficiency is far lower than Huffman.

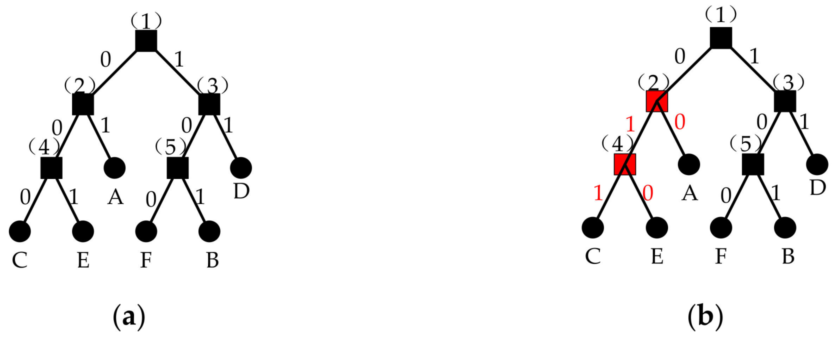

Figure 9. (**a**) Basic Huffman tree; (**b**) Mutated Huffman tree by changing the label-pairs of inner-nodes (2) (4).

Therefore, the key to improve fusion efficiency is to reduce these additional computational losses. In fact, as can be seen from Figure 10, the process from *T*0 to *T*1 to *T*2 to *T*3 has the same result as the process from *T*0 directly to *T*3, depending only on the value of the key-stream, not the times of mutations. So *T*1, *T*2, and *T*3 can be obtained directly from *T*0, as shown in Figure 10. That is to say, as long as the original code is through the key temporary generation of new code, HTM can achieve the same effect of the compression and encryption.

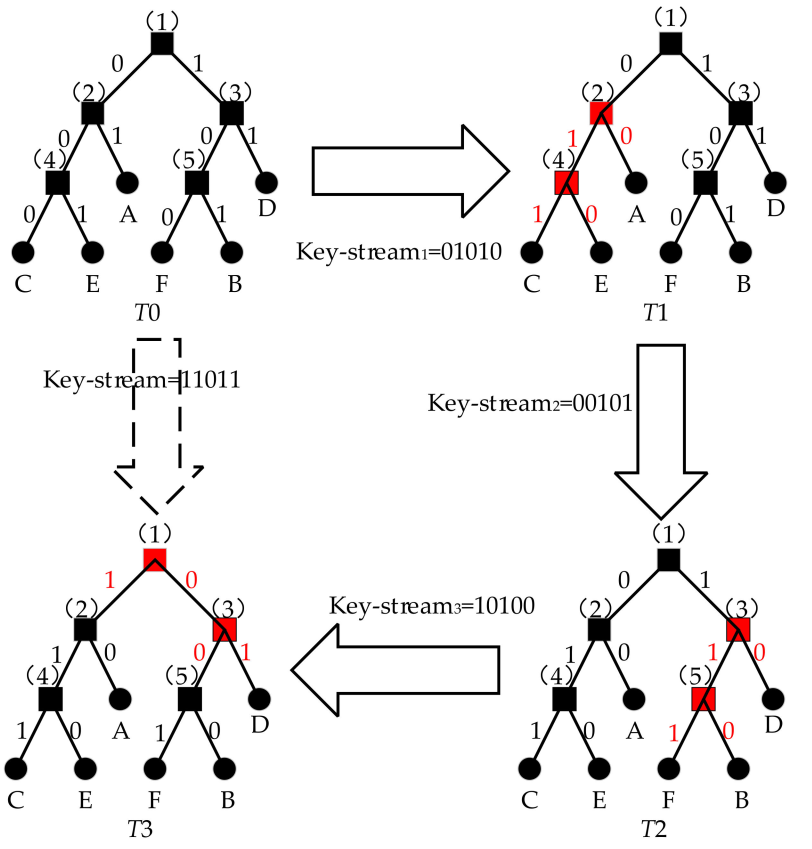

Figure 10. The process of mutation of Huffman tree.

So, a modified HTM technique is proposed to reduce the operation cost of the Huffman Tree update: the mutation of the inner-node from basic Huffman tree instead from the mutated Huffman tree. The modified technique removes the Huffman tree inner-node coding update operation in each mutation process, which will save N-1 update operations each.

A parallel mutation method based on the basic Huffman tree is proposed for operational efficiency by reducing the computational cost of the merging process and employing a parallel architecture.

3.7. Splice Module

The variable length encoding of Huffman code brings about the compression of data, but it also brings the problems for hardware implementation. After encoding, the code word is of variable length. Since the width of the data bus is fixed, the code stream composed of variable length code sign needs to be divided and packaged into fixed-length code and output to the data bus.

First, the code word and code length are obtained from the foregoing unit and are latched into registers, respectively. The bucket shift register concatenates and locks the contents of the register. When the register is filled, the data in it is output to the data bus, and the bucket shift register data move in. The remaining codewords are thus joined to the new ones through the bucket shift register.

During the process, the variable-length codeword stream is divided into fixed-length codeword streams and then transferred to a fixed-width data bus.

3.8. Parallel Architecture

As shown in Figures 6 and 7, the PUFs key generation can be performed in parallel with the module statistic and the basic Huffman tree construction. The probability statistics of symbol and the construction of Huffman tree, are the most time-consuming operation in the whole Huffman algorithm. Even with the most efficient heap sorting algorithm, the time complexity reaches o($nlog_2n$).

The process of generating the random number in PUFs is also a time-consumed and independent to Huffman coding. The blocks are designed in parallel to make the execution time reduce from $t_l + t_2$ to max $\{t_1, t_2\}$, which will greatly improve the execution efficiency.

4. Experiments and Performance Evaluation

In this section, to prove that our scheme is practical and effective, we first introduce the features of our implemented BR-PUF, and then show the PUFs key is reliable to be taken into the practical use. Finally, we constructed a file protection system to evaluate the compression performance of our proposed JCAE scheme.

4.1. Performance of Implemented PUFs

In this part, the performance of PUFs instanced on our hardware platform are evaluated by the method proposed in work [31]. Since the PUFs circuit is used to generate PUFs keys that play an important role of mutation the Huffman Tree, it is necessary to prove the performance of the PUFs inserted in our JCAE scheme is satisfactory. In this paper, we instance the BR-PUF on our hardware platform as shown in Figure 11.

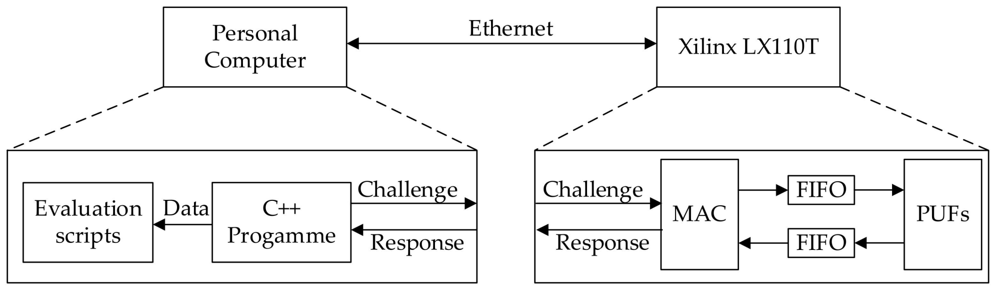

Figure 11. Schemes follow the same formatting.

The test system was constructed as shown in Figure 11 with a Xilinx LX110T FPGA and a personal computer. The communication channel between two devices is constructed by the Ethernet. The FPGA implemented 16 128-bits BR-PUF to generate responses. The personal computer sends challenges, receives the corresponding responses, and uses evaluation scripts to evaluate the performance of PUFs.

The performance of PUFs is evaluated from the following three aspects.

1. Reliability

The reliability of the PUFs circuit reflects the ability of reproducing the same response according to a repeated challenge. When working on a noisy working condition, there will be the noisy in the response to influence the reliability of the PUFs circuit. The noisy and reliability are calculated as the Equations (2)–(4):

$$FHD(X,Y) = \frac{HD(X,Y)}{m} \tag{2}$$

$$L_{noise} = \frac{1}{n}\sum_{i=1}^{n} FHD(R_x, R_{x,y}) \times 100\% \tag{3}$$

$$\mathrm{Re}liablity = 100\% - L_{noise} \tag{4}$$

where R_x is a m-bits reference response, $R_{x,y}$ is the y-th sampling of R_x, *HD(X,Y)* is the hamming distance between *X,Y*, and each PUFs challenge has been repeated n times. We tested the reliability on seven different temperatures ranging from 0 °C to 60 °C. For each PUFs circuit, we collected 81,920 responses at each temperature, and the responses collected at 30 °C are used as reference responses. The noise L_{noise} in response is close to a state of normal distribution, where the expected value is 0.0494, the variance is 0.0131, and the biggest value of noise is less than 0.09. According to Equation (6), the reliability is 0.9506.

2. Randomness

The randomness of PUFs describes the randomness of the responses generated by the PUFs circuit. In other words, the uniformity represents the possibility of "0" or "1" occurs in every bit. Therefore, the ideal randomness of PUFs is 50%. If the randomness is close to 50%, it will become difficult for the adversary to guess the actual response of PUFs. The randomness of PUFs is calculated as Equation (5):

$$Randomness = -p \times \log_2^p - (1-p) \times \log_2^{(1-p)} \tag{5}$$

where p is the proportion of the bit "1" in the response. The experiment results show that the randomness of these 16 implemented BR-PUF is between 0.9261 and 0.9999, and the average value is 0.9865.

3. Uniqueness

The uniqueness is used to evaluate the difference of the response between two PUFs circuits, when they work on the same condition and receive the same challenge. The ideal value of uniqueness is 50%, and it means half of the bits in response from two PUFs circuits are different. The uniqueness of PUFs circuit is computed as Equation (6):

$$Uniqueness = \frac{2}{k(k-1)}\sum_{i=1}^{k-1}\sum_{j=i+1}^{k} FHD \tag{6}$$

where k is the number of PUFs circuits. The experiments shows that the uniqueness of these 16 BR-PUF obeys normal distribution, where the expected value is 0.4892 and the variance is 0.322. Therefore, the uniqueness of our implementation is 0.4892.

4.2. Performance of Generated Key-Stream from PUFs

The responses of the PUFs circuit are used to generate the secret key to mutate the Huffman tree. Thus, it is necessary to evaluate the reliability of the PUFs key and check whether the responses produced by the PUFs circuit meet the requirements of acting as the random number.

The fuzzy extractor was implemented by Reed-Muller code (16, 5, 8) and Repetition code (16, 1, 16) and the probability of generating the incorrect PUFs key is reduced to 1.7257×10^{-12}, which is a sufficient small value. Therefore, the PUFs key is practical to be used in our scheme. Moreover, NIST random tests [32] are carried on the generated responses. The results are listed in Table 2 and pass all the 15 checks. As a result, the responses from the PUFs circuit are suitable to be used for the key generation and the PUFs key is reliable to be integrated into our JCAE scheme.

Table 2. Results of NIST random check.

Content	*p*-Value	Conclusion
Frequency	0.699313	accept
BlockFrequency	0.145326	accept
CumulativeSums	0.791242	accept
Runs	0.534146	accept
LongestRun	0.964295	accept
Rank	0.851383	accept
FFT	0.474986	accept
NonOverlappingTemplate	0.497054	accept
OverlappingTemplate	0.019188	accept
Universal	0.289667	accept
ApproximateEntropy	0.554420	accept
RandomExcursions	0.364111	accept
RandomExcursionsVariant	0.407702	accept
Serial	0.226137	accept
LinearComplexity	0.911413	accept

4.3. Performance of Proposed JCAE Scheme

To further evaluate the actual performance of our JCAE scheme, we implemented our JCAE hardware circuit and constructed a file protection system for cloud server. The proof-of-concept prototype of the file protection system is shown in Figure 12. The system mainly consists of three parts, including the cloud server, the hardware implementation of our proposed JCAE scheme, and the communication channel.

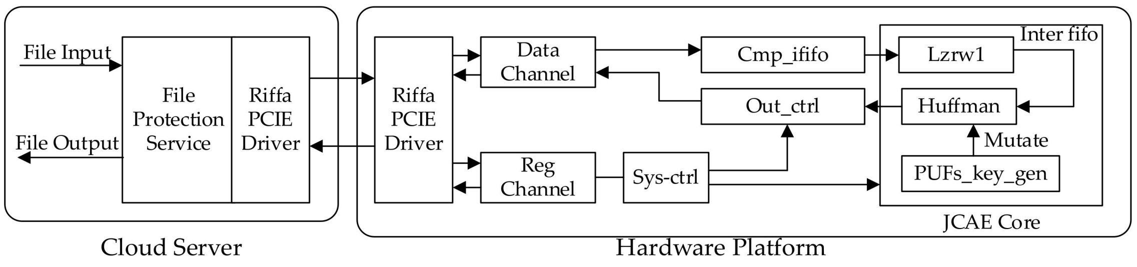

Figure 12. Architecture of File Protection System.

Cloud server: The cloud server that is responsible of storing the huge volume of data was implemented on the Fiber Home server Fit Server R2200 V5. To protect the sensitive data, the file protection service will send the data stream of the input file to the hardware circuit for further processing. The ciphertext that is compressed and encrypted by the hardware circuit will be sent back and stored on the cloud server.

Hardware platform: The hardware circuit of our proposed JCAE scheme was instanced on the FPGA Xilinx LX110T, which is shown in the right side of Figure 12. The Data Channel is used to exchange the data stream with the help of PCIE and the Reg Channel receives the control signal from the cloud server to trigger the system control module (sys_ctrl). The

JCAE core is composed of two connected compression modules, which implement LZWR1 (Variation of LZW) compression and Huffman lossless compression respectively. It is noted that the introducing of LZWR1 is to a common method to improve the compression rate and make our JCAE scheme more practical to be used in the file protection system. This method maps the entropy source of the input file to a proper range, where the Huffman encoding executes efficiently. The PUFs key generation (PUFs_key_gen) module generates the PUFs key and mutates the Huffman tree to jointly compress and encrypt the data. The encrypted data will be transmitted to the PCIE through the output control module (out_ctrl).

Communication channel: The communication channel is constructed by PCIE to share the data between the cloud server and the hardware platform and achieve a high transmission rate. We used the open-source project to implement the PCIE module on both cloud server and hardware platform.

Twenty-nine various files chosen from Calgary and Canterbury corpus were input into our file protection system to evaluate the performance of our JCAE scheme. The test results are recorded in Table 3 from three aspects, including the operation time, the throughput, and the compression ratio.

Table 3. Test results.

File Name	Input File (Bytes)	Output File (Bytes)	Time (ms)	Throughput (Mbps)	Compression Ratio
book1	768,771	530,298	14.18	423.5	68.98%
book2	610,856	365,215	6.63	720.24	59.79%
geo	102,400	94,927	1.54	519.8	92.70%
news	377,109	233,104	4.2	700.78	61.81%
obj1	21,504	13,774	0.43	391.65	64.05%
obj2	246,814	132,699	4.74	406.61	53.76%
paper1	53,161	31,275	0.82	504.05	58.83%
paper2	82,199	50,536	2.33	275.98	61.48%
paper3	46,526	29,536	0.62	588.24	63.48%
paper4	13,286	8218	0.37	279.31	61.85%
paper5	11,954	7305	0.29	323.08	61.11%
paper6	38,105	22,007	0.67	446.33	57.75%
pic	513,216	131,602	5.87	682.92	25.64%
progc	39,611	22,025	0.54	568.92	55.60%
progl	71,646	31,684	0.86	647.83	44.22%
progp	49,379	21,528	0.62	623.08	43.60%
trans	93,695	44,218	1.01	722.64	47.19%
alice29.txt	152,089	94,038	1.87	635.77	61.83%
asyoulik.txt	125,179	80,728	2.99	326.55	64.49%
cp.html	24,603	12,899	0.44	441.8	52.43%
fields.c	11,150	5176	0.31	283.85	46.42%
grammar.lsp	3721	1875	0.23	128.17	50.39%
kennedy.xls	1,029,744	337,936	13.13	612.64	32.82%
lcet10.txt	426,754	255,553	7.32	455.6	59.88%
plrabn12.txt	481,861	330,812	5.28	713.24	68.65%
ptt5	513,216	131,602	8.11	494.34	25.64%
sum	38,240	23,258	1.06	281.82	60.82%
xargs.1	4227	2489	0.56	58.66	58.88%

The throughput of our proposed scheme is efficient. As can be seen from Table 3, the throughput for most of the files precedes 400 Mbps, which is an efficient value. However, the throughput for some small files is fluctuant. The file is split into 16 KB data blocks to be transmitted between the cloud server and the hardware platform. If the file size is small and no more than 100 KB, serval data blocks cannot reflect the working efficiency of the compression pipeline. Therefore, the throughput for big files proves the operational efficiency of our scheme.

The compression ratio of our scheme is reasonable and satisfactory. The compression ratio is defined as the following Equation (7):

$$R = \frac{Size\_of\_Final\_data}{Size\_of\_Original\_data} \times 100\% \tag{7}$$

The compression ratio for most of the files in our test is variable and approximately around 60%. The difference is mainly caused by the various entropy value in different test files. In other word, repeated data or data that can be compressed will not be the same in each file. In general, the compression ratio of our scheme is proved to achieve a good value.

Although the operation time is closely related to the size of the input file, the two of the biggest test files book1 and kennedy.xls only take 14.18 ms and 13.13 ms, respectively, to finish the joint compression and encryption. Therefore, the operation times is efficient to prove that our scheme is practical to be used in the file protection system.

5. Security Analysis

In this section, we analyze the security of scheme based on four different attack models, including one physical attack model and three traditional attack models, to further prove that our scheme is effective to enhance the security level of existing date protection methods.

5.1. Physical and Cloning Attacks

With the abilities of performing physical and cloning attacks, the adversary would derive the secret key from the cloud server by methods of power analysis. The attacker may use the physical attack method of power analysis [33] to derive the secret key stored in devices. When the PUFs circuit is integrate into our JCAE scheme, there is no need to store any secret key. Moreover, the PUFs circuit is unclonable as evaluated in Section 4.1, and the attacker cannot clone the circuit to acquire the real PUFs response. The attacker can only guess the correct secret key with 128-bit length, the probability of which is sufficiently small. The PUFs key that is derived from the hardware circuit will be used to mutate the Huffman tree to realize the function of encryption in our proposed scheme. Therefore, it is hard for the adversary to obtain the right PUFs key and build the corresponding Huffman tree to decrypt the ciphertext. In general, our proposed JCAE scheme is resistant to physical and cloning attacks.

5.2. Ciphertext-Only Attack

The ciphertext-only attack is the least favorable for the adversary among these three traditional attack models. Under this model, the adversary can only obtain the ciphertext of the protected sensitive data. The ciphertext encoded by the methods of the mutated Huffman Tree has the good randomness with statistical irregularities. Thus, the adversary must perform the exhaustive key research on a large key space. As is described in section three, the key is the response of the PUFs circuit, which has the bit length of 128. As a result, the key space of our scheme is 2^{128} and makes it impossible for the adversary to operate the exhaustive key research to construct the right Huffman tree. In general, our scheme is resistant to ciphertext-only attack.

5.3. Know-Plaintext Attack

The know-plaintext attack model is stronger than the ciphertext-only attack. In this attack model, the adversary can obtain a number of plaintext/ciphertext pairs by leakage of insider information or guessing. The goal of the adversary is to rebuild the mutated Huffman tree that is used to encrypt the plaintext. Thus, it is necessary to find out the corresponding relationship between the output bits and symbols in the input data. However, the synchronization problem is extremely difficult for the adversary to solve, since the length of each encoded/encrypted symbol has at least two different possibilities.

Then, the difficulty is evaluated by calculating the number of the different synchronization ways. It is assumed in this situation that the input data has M = 32N symbols and each symbol has two possible bit lengths of l_i and l_j. As a result, the bit length of the output is shown in Equation (8):

$$L + \sum_{i=0}^{M-1} l_i, L \in (0, M) \tag{8}$$

The possible ways of synchronization are calculated as C_M^L, where L is the number of symbols with l_i bit length. If the M is set to 100 and SP is the number of the synchronization patterns, when L increases from 1 to 99, the logarithm number log_2SP will increases from 6.67 to 96.35, and reaches the peak when L equals 50. In conclusion, if we choose the parameters properly, the number of the synchronization patterns is close to 2^{100}, making the adversary hard to solve the synchronization problem as shown in Figure 13. Therefore, our scheme is resistant to know-plaintext attack.

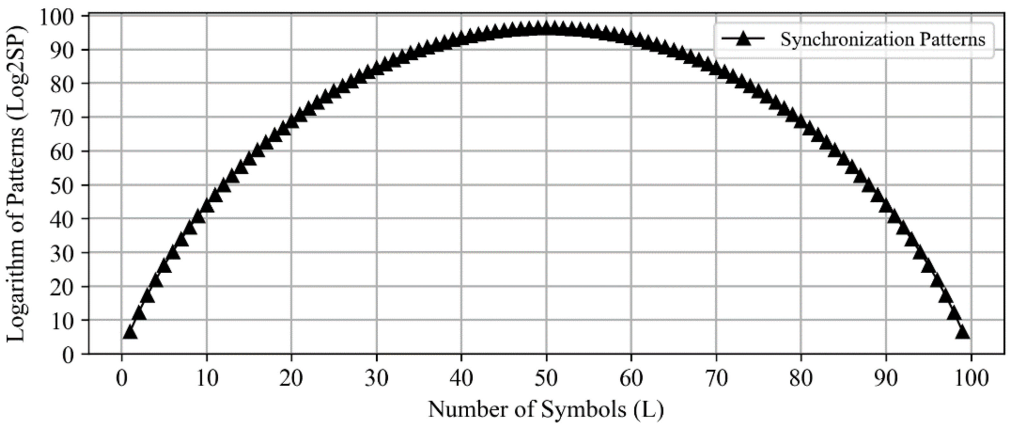

Figure 13. Number of the Synchronization patterns.

5.4. Chosen-Plaintext Attack

The adversary has the most powerful capabilities in the chosen-plaintext attack model. In this model, the adversary can send random symbol to the encryption system and get the corresponding output bit stream, with the purpose of deriving the secret key to break the security of the encryption system. If the adversary sends the plaintext byte by byte and analyzes the output bit stream continuously, our scheme is vulnerable to this attack model. However, to effectively resist this kind of attack, our scheme accumulates the input symbols to form a big chunk and outputs the corresponding bit stream at one time, as described section three. The size of the data chunk is closed related to the length of the vector P. Although this accumulation method will bring the time delay, the performance evaluation mentioned above has proven the influence is limited.

Moreover, it is also a common attack method for the adversary to launch the differential attack by resetting the encryption system in this chosen-plaintext model. For example, the adversary would first insert one symbol X at the beginning of a whole symbol chunk (0, 0, ,0, 0) to construct a new symbol chunk, as (X, 0, ,0, 0) in this attack method. Then, the encryption system will encode/encrypt these two chunks and outputs two bitstream with different length x, y respectively. By analyzing the result of ∣x-y∣ with the help of the basic Huffman table, the adversary can obtain the real length of the encryption result of X. If this process is repeated many times, the synchronization method mentioned in the above attack model would be solved. In general, to withstand the chosen-plaintext attack, our JCAE scheme should encode/encrypt a whole chunk at one time and restrict the reset request.

In conclusion, although the security mechanism provided by our scheme is not as strong as the common encryption algorithms, the physically secure feature is valuable, and actually improves the security level of data storage.

6. Conclusions and Future Research

To enhance the data storage method of cloud servers with the high operational efficiency and robust security, we proposed a Huffman-based JCAE for the secure data storage using PUFs. Firstly, the BR-PUF circuit was instanced to make the proposed scheme resist physical and cloning attacks. Secondly, the modified mutated Huffman tree technique was proposed to reduce the computational cost. A parallel architecture was designed to in the hardware platform to reduce the execution time. Lastly, the LZW algorithm was combined with the Huffman algorithm to get a satisfactory compression ratio, which made it generic and practical to adapt to the existing data storage method.

Finally, the proposed JCAE scheme was implemented on the FPGA Xilinx LX110T and then integrated into a proof of the concept file protection system. The results showed that the PUFs in the scheme are qualified with reliability value 0.9506, randomness value 0.9865, and uniqueness value 0.4892, and the key-stream generated by PUFs passed NIST random tests. Further tests with Calgary and Canterbury corpus were run in the file protection system for cloud server. The average throughput is 473 Mbps and average compression ratio is 55.86%. Additionally, the security was analyzed based on Physical and Cloning Attacks, Ciphertext-Only Attack, Know-plaintext Attack, and Chosen-Plaintext Attack, which further proved the proposed scheme enhances the security level of existing date protection methods.

However, we must point out the limitations of our scheme: the BR-PUF adopted in this paper can only be implemented on the application architecture with strong PUFs structure, so it is not suitable for the application architecture with weak PUFs structure. How to design a JCAE scheme using weak PUFs is a future plan to be solved. It is suggested to make compound cryptosystem with weak PUFs, and CRP extracted from weak PUFs is taken as the seed to generate public-private key pair.

Furthermore, the higher throughput and security have always been the constant research trend of JCAE. How to integrate the compression algorithm and encryption algorithm as far as possible under the premise of maintaining the compression ratio is the main development direction in the future.

Author Contributions: Conceptualization, Y.L. and B.L.; methodology, Y.L.; software, X.Z.; validation, Y.L., Y.Z. and X.Z.; writing—original draft preparation, Y.L.; writing—review and editing, Y.L., B.L. and Y.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the national natural science foundation of China (No. 61571116), the basic research (exploration) of science and technology in Shenzhen (JCYJ20170817115538543), and the basic research (layout) of science and technology in Shenzhen (JCYJ20170817115500476).

Data Availability Statement: Publicly available datasets were analyzed in this study. This data can be found here: [https://corpus.canterbury.ac.nz/descriptions/].

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

References

1. Kumari, S.; Karuppiah, M.; Das, A.K.; Li, X.; Wu, F.; Kumar, N. A secure authentication scheme based on elliptic curve cryptography for IoT and cloud servers. *J. Supercomput.* **2018**, *74*, 6428–6453. [CrossRef]
2. Jiang, D.; Wang, Y.; Lv, Z.; Wang, W.; Wang, H. An Energy-Efficient Networking Approach in Cloud Services for IIoT Networks. *IEEEJ. Sel. Areas Commun.* **2020**, *38*, 928–941. [CrossRef]
3. Park, J.H.; Hassan, H. Advanced algorithms and applications for IoT cloud computing convergence. *J. Parallel Distrib. Comput.* **2018**, *118*, 265–266. [CrossRef]
4. Hammoudi, S.; Aliouat, Z.; Harous, S. Challenges and research directions for Internet of Things. *Telecommun. Syst.* **2018**, *67*, 367–385. [CrossRef]
5. Mohammed, R.S.; Mohammed, A.H.; Abbas, F.N. Security and Privacy in the Internet of Things (IoT): Survey. In Proceedings of the 2019 2nd International Conference on Electrical, Communication, Computer Power and Control Engineering (ICECCPCE), Mosul, Iraq, 13–14 February 2019; pp. 204–208.
6. Cui, H.; Deng, R.H.; Qin, B.; Weng, J. Key regeneration-free ciphertext-policy attribute-based encryption and its application. *Inf. Sci.* **2020**, *517*, 217–229. [CrossRef]

7. Elhabob, R.; Zhao, Y.; Eltayieb, N.; Abdelgader, A.M.S.; Xiong, H. Identity-based encryption with authorized equivalence test for cloud-assisted IoT. *Clust. Comput.* **2020**, *23*, 1085–1101. [CrossRef]
8. Manikandan, G.; Perumal, R. Symmetric cryptography for secure communication in IoT. *Mater. Today: Proc.* **2020**. [CrossRef]
9. Wu, C.P.; Kuo, C.C. Design of integrated multimedia compression and encryption systems. *IEEE Trans. Multimed.* **2005**, *7*, 828–839. [CrossRef]
10. Hermassi, H.; Rhouma, R.; Belghith, S. Joint compression and encryption using chaotically mutated Huffman trees. *Commun. Nonlinear Sci. Numer. Simul.* **2010**, *15*, 2987–2999. [CrossRef]
11. Jiang, J.; Pan, J.-S.; Tang, L.; Chen, C.-C. Enhanced Huffman Coding with Encryption for Wireless Data Broadcasting System. In Proceedings of the 2012 International Symposium on Computer, Consumer and Control, Taichung, Taiwan, 4–6 June 2012. [CrossRef]
12. Song, Y.; Zhu, Z.; Zhang, W.; Yu, H. Efficient protection using chaos for Context-Adaptive Binary Arithmetic Coding in H.264/Advanced Video Coding. *Multimed. Tools Appl.* **2019**, *78*, 18967–18994. [CrossRef]
13. Klein, S.T.; Shapira, D. Integrated encryption in dynamic arithmetic compression. *Inf. Comput.* **2020**, 104617. [CrossRef]
14. Yu, C.; Li, H.; Wang, X. File compression and encryption based on LLS and arithmetic coding. *IOP Conf. Ser. Mater. Sci. Eng.* **2018**, *322*, 062011. [CrossRef]
15. Zhu, Z.; Tang, Y.; Liu, Q.; Zhang, W.; Yu, H. A Chaos-based Joint Compression and Encryption Scheme Using Mutated Adaptive Huffman Tree. In Proceedings of the 2012 Fifth International Workshop on Chaos-fractals Theories and Applications, Dalian, China, 18–21 October 2012; pp. 212–216.
16. Tsai, C.-J.; Wang, H.-C.; Wu, J.-L. Three Techniques for Enhancing Chaos-Based Joint Compression and Encryption Schemes. *Entropy* **2019**, *21*, 40. [CrossRef]
17. Jakimoski, G.; Subbalakshmi, K.P. Cryptanalysis of Some Multimedia Encryption Schemes. *IEEE Trans. Multimed.* **2008**, *10*, 330–338. [CrossRef]
18. Liu, B.; Chen, Z.; Zhang, Y.; Xiong, L.; Yang, X.; Chen, S.; Li, B. A New Group-to-Group Authentication Scheme Based on PUFs and Blockchain. In Proceedings of the 2019 IEEE 4th International Conference on Signal and Image Processing (ICSIP), Wuxi, China, 19–21 July 2019; pp. 279–283.
19. Chen, S.; Li, B.; Zhou, C. FPGA implementation of SRAM PUFs based cryptographically secure pseudo-random number generator. *Microprocess. Microsyst.* **2018**, 59. [CrossRef]
20. Zhang, Y.; Li, B.; Liu, B.; Wu, J.; Wang, Y.; Yang, X. An Attribute-Based Collaborative Access Control Scheme Using Blockchain for IoT Devices. *Electronics* **2020**, *9*, 285. [CrossRef]
21. McIntosh, T.; Watters, P.; Kayes, A.S.M.; Ng, A.; Chen, Y.-P. Enforcing situation-aware access control to build malware-resilient file systems. *Future Gener. Comput. Syst.* **2020**, *115*, 568–582. [CrossRef]
22. Kayes, A.S.M.; Rahayu, W.; Dillon, T.; Chang, E.; Han, J. Context-Aware Access Control with Imprecise Context Characterization for Cloud-Based Data Resources. *Future Gener. Comput. Syst.* **2018**, 93. [CrossRef]
23. Sharma, S.; Chang, V.; Tim, U.; Wong, J.; Gadia, S. Cloud and IoT-based emerging services systems. *Clust. Comput.* **2019**, 22. [CrossRef]
24. Xue, Y.; Xue, K.; Gai, N.; Jianan, H.; Wei, D.; Hong, P. An Attribute-Based Controlled Collaborative Access Control Scheme for Public Cloud Storage. *IEEE Trans. Inf. Forensics Secur.* **2019**. [CrossRef]
25. Balduzzi, M.; Zaddach, J.; Balzarotti, D.; Kirda, E.; Loureiro, S. A security analysis of amazon's elastic compute cloud service. *Proc. Acm Symp. Appl. Comput.* **2012**. [CrossRef]
26. Somorovsky, J.; Heiderich, M.; Jensen, M.; Schwenk, J.; Gruschka, N.; Lo Iacono, L. *All Your Clouds Are Belong to Us—Security Analysis of Cloud Management Interfaces*; ACM: Chicago, IL, USA, 2011; pp. 3–14.
27. Chen, Q.; Csaba, G.; Lugli, P.; Schlichtmann, U.; Rührmair, U. *The Bistable Ring PUF: A New Architecture for Strong Physical Unclonable Functions*; IEEE: San Diego, CA, USA, 2011; pp. 134–141.
28. Garg, A.; Lee, Z.C.; Lu, L.; Kim, T.T.-H. Improving uniformity and reliability of SRAM PUFs utilizing device aging phenomenon for unique identifier generation. *Microelectron. J.* **2019**, *90*, 29–38. [CrossRef]
29. Gu, C.; Liu, W.; Cui, Y.; Hanley, N.; Oneill, M.; Lombardi, F. A Flip-Flop Based Arbiter Physical Unclonable Function (APUF) Design with High Entropy and Uniqueness for FPGA Implementation. *IEEE Trans. Emerg. Top. Comput.* **2019**. [CrossRef]
30. Namburi, S.; Rao, P.; Muvva, P.; Muvva, P.K.; K, S.K. An Efficient Method to Reduce LZW Algorithm Ouput Code Length. *Int. J. Eng. Appl. Sci. Technol.* **2020**, *4*, 302–304. [CrossRef]
31. Maiti, A.; Gunreddy, V.; Schaumont, P. A Systematic Method to Evaluate and Compare the Performance of Physical Unclonable Functions. *IACR Cryptol. Eprint Arch.* **2011**, *2011*, 657. [CrossRef]
32. Rukhin, A.; Soto, J.; Nechvatal, J.; Smid, M.; Barker, E.; Leigh, S.; Levenson, M.; Vangel, M.; Banks, D.; Heckert, A.; et al. NIST Special Publication 800-22: A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications. *NIST Spec. Publ. 800-22* **2010**. [CrossRef]
33. Messerges, T.S.; Dabbish, E.A.; Sloan, R. Examining Smartcard Security under the Threat of Power Analysis Attacks. *IEEE Trans. Comput.* **2002**, *51*, 541–552. [CrossRef]