IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received February 28, 2022, accepted March 17, 2022, date of publication March 25, 2022, date of current version April 6, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3162238

An Automatically Privacy Protection Solution for Implementing the Right to Be Forgotten in Embedded System

YANAN ZHAO[1], NONG SI[1], (Member, IEEE), YU SUN[1], XIN GAO[1], HAOPENG TONG[1], AND GENG YUAN[2]

[1]Faculty of Information Technology, Beijing University of Technology, Chaoyang, Beijing 100124, China
[2]Faculty of Natural Science, Kristianstad University, 291 88 Kristianstad, Sweden

Corresponding author: Yu Sun (respectprivacy@yeah.net)

This work was supported in part by the Industry-University Collaborative Foundation of Ministry of Education of China, and in part by Huawei under Grant 201902146003.

ABSTRACT Towards the massive amount of data generated in our daily work and life, embedded systems, with economical but powerful storage and computing resources, are inevitably becoming the most suitable platform for the Edge Computing for the Internet of Things. However, embedded system servers may also threaten individuals by storing individuals' private data for years. This paper proposes a Resilient Tag-based Privacy Protection (RTPP) scheme for embedded systems. Specifically, to protect the privacy against the hackers and other non-users, we employ a pseudo-random number encryption technique with the chaos-based principle so that the third party cannot easily steal the private data and reduce the risk of personal privacy leakage. To protect the individuals' interests, we propose a new approach to controlling the life cycle table of data to enable individuals themselves the flexibility to control the life cycle of private data. Unlike existing data lifetime management methods, the RTPP can support the retrieval of tags in the data life cycle table to control the corresponding privacy while automatically adding or removing tags. Our system automatically adjusted the survival period of private data in the life cycle table through the change of leaf weights, controlled the charge movement on the surface of flash memory, and finally achieved the resilient adjustment process of the life cycle of private data in the embedded system. The security proof and performance evaluation show that the proposed RTPP scheme is provable secure in the automatic privacy lifecycle tuning model for embedded systems and efficient in practice.

INDEX TERMS Huffman coding, information security, chaotic mapping, flash memory, data lifecycle.

I. INTRODUCTION

Automatically and opportunely deleting the correct personal data in an embedded system is challenging to protect privacy. As the European Union's General Data Protection Regulation (GDPR) [1] went into effect on May 25, 2018, and the California Consumer Privacy Act (CCPA) [2] became effective on January 1, 2020, these laws contribute the rise of attention to individuals' private data using, protecting, deleting and forgetting. While website visitors choose to allow cookies or upload personal data to the websites, the service provider will automatically record our preferences, individual private data on their databases for years. Such activities increase the security risk of violating personal privacy under the above laws. However, people have the right to ask the data owner to delete personal information from any databases according to their requirements, fulfilling the legal "right to be forgotten" [3].

Except on the internet, due to the worldwide epidemic prevention and control, a large amount of personal information is collected by various devices, which poses a significant security risk to individuals' privacy. Traditionally, there are two ways to prevent privacy leakage, one is to enhance the security of encryption algorithms in software to protect sensitive data, and the other is to remove private data directly from the hardware. Since most encryptions can be decrypted on purpose with adequate time, it is more thorough

The associate editor coordinating the review of this manuscript and approving it for publication was Jiafeng Xie.

in removing private data directly from the hardware. Therefore, research on the automatic and complete removal of personal data from hardware has become a hot topic in recent years.

In this work, we designed and developed Resilient Tag-based Privacy Protection (RTPP) scheme. In the RTPP, much personal private data is sensitive, so the first thing to consider is private data encryption. We propose and evaluate an encryption method based on chaos theory for pseudo-random number generators. Since chaotic systems are susceptible to initial states and complex dynamic behavior, chaotic systems do not follow the probability statistics in the distribution. The proposed random sequence can provide a good randomness seed for the pseudo-random number generator, making the encryption system we design challenging to be broken for higher security. Secondly, we designed the Data Label Life Cycle Table (DLLCT). It allows dynamic and flexible control of the data lifecycle, enabling users to manage their private data more efficiently and conveniently.

The rest of this paper is structured as follows: Section II reviews the existing methods for implementing ''auto-forgotten'' for embedded systems and cryptographic algorithms based on chaos theory. Section III describes the design of the proposed pseudo-random number generator based on chaos theory. Section IV presents the RTPP scheme. Section V presents the performance and security analysis of the implemented algorithm. Section VI summarizes the entire paper and provides suggestions for future work.

II. RELATED WORKS

This section presents related existing methods for embedded automatic being forgotten and compares them intuitively. In addition, we investigate the suitability of chaos theory for improving encryption algorithms used for pseudo-random numbers.

A. THE EXISTING METHODS FOR IMPLEMENTING "AUTO-FORGOTTEN" FOR EMBEDDED SYSTEMS

Many automatic forgotten methods have been proposed that are suitable for implementing the protection of personal privacy data in embedded systems. The hardware implementations of these approaches are usually analyzed based on the complexity of privacy data storage using a combination of spatial complexity and temporal complexity.

Tanakamaru *et al.* proposed the PP-SSS System in 2015 [4], which automatically destroys personal private data by setting the exact life spans for the different physical storage units. Data destruction is performed by consciously writing deliberate errors so that the error correction system cannot identify private data outside the expected life span. Compared to traditional data deletion methods, privacy-preserving solid-state storage systems remove personal privacy more directly from the source than hiding data from the user. However, the effectiveness of this system is limited to compressed data. It is also not suitable for the long-term storage of private personal data, as the data life cycle is different. Yamazawa *et al.* in 2016 used precise ECC and shredding techniques to precisely control the storage lifetime of private data in hardware [5]. Suzuki *et al.*, in 2019, designed the PDLCS [6]. PDLCS, in comparison to PP-SSS, adds the process of In-3D vertical cell processing, where the lateral charge migration in 3D NAND flash controls the lifetime of the data, which provides a more efficient guarantee for a longer or shorter private data lifecycle.

However, this system also has drawbacks. Firstly, it only performs simple encryption during the data processing process of the original private data, which can easily lead to privacy leakage. Secondly, it does not propose an exact data lifecycle management scheme. Multiple private data are processed one by one, increasing processing time, consuming embedded systems, processing process's complexity, and depleting battery life. Therefore, we focus on the issue that the hardware can automatically adjust the lifecycle of private data without decreasing the security level of private data.

B. CHAOS-BASED ENCRYPTION ALGORITHM

The existing chaotic cryptography is achieved in two steps: first, a pseudo-random key stream is generated using a chaotic system, and the plaintext is encrypted using the generated key stream, called stream-based chaotic encryption [7]. Second, the ciphertext is obtained by multiple iterations (or reverse iterations) using the plaintext (or key) as the initial condition (or control parameter) to achieve encryption. This method belongs to block-based chaotic encryption, widely used for traditional packet encryption such as DES and AES [8]. In chaotic encryption algorithms, chaotic mapping is often referred to as the core component of the encryption process, which generates many pseudo-random sequences [9]. The general idea of designing chaos-based ciphers is to use the sequences in chaotic mappings to perform cryptographic operations on-target messages [10]. Therefore, to improve the security of cryptosystems, chaotic mappings need to be continuously optimized. In 2011, Cao *et al.* improved the complexity of chaotic mappings by changing the parameters [11]. In 2019, Peng *et al.* added the quantum chaos and PWLCM chaotic mapping into a new method of S-box design, which significantly improved the security performance of the cryptography [12]. In 2020, Patel *et al.* proposed an improved 3D chaos logistic map encryption algorithm, which makes the encryption algorithm strong [13]. Currently, the construction of hash function based on chaotic mapping is a research direction in chaotic cryptography, which uses the sensitivity of initial values and pseudo-randomness inherent in chaotic systems to generate hash values. These Hash values are used as seeds for pseudo-random number ciphers, which finally undergo several chaotic iterations to generate unpredictable random keys. Among such studies, in 2016, Li *et al.* proposed the construction of a one-way hash function based on a sequence design with double perturbations of spacetime chaos [14]. In 2015, Teh *et al.* proposed the construction of a hash function based on chaotic logic equations [15]. Meanwhile, it is proved that a single low-dimensional chaotic system is more vulnerable to attacks. In contrast, a high-dimensional chaotic system can improve security but reduce the speed of cryptographic operations.

Therefore, considering the above problems, a MAC pseudo-random function generator based on segmented logistic chaotic mapping for RTPP system is designed in this paper from the viewpoint of efficiency and security to complete the storage encryption of private data.

III. THE PROPOSED ALGORITHM: MODIFIED HMAC (CHMAC)

The security of storing private data is as important as the memory usage in the embedded system to implement the automatic forgotten scheme of private data. Since most MAC-based pseudo-random number generators are constructed using the MAC algorithm [16] with embedded hash functions (HMAC) [17], in this study, cryptographers aim to design an algorithm that is more resistant to attacks than HMAC. Therefore, we propose a Chaos-based HMAC (CHMAC) algorithm in this subsection, and further details of the HMAC algorithm and the CHMAC algorithm are presented.

A. HMAC ALGORITHM

The HMAC algorithm uses the underlying Hash function with the key to complete the encryption process, which is defined as follows [2]:

$$\text{HMAC}(K, M) = H[(K^+ \oplus \text{opad})||H[(K^+ \oplus \text{ipad})||M]] \tag{1}$$

where K is the key shared by both communication parties, M is the message to be verified. H is the embedded Hash function. "$\oplus$" means "bitwise iso-or" operation. "$||$" means "or" operation. When the length of the key K is less than the number of bits b contained in each group of the Hash, the length of K and b are the same by adding 0 to the end of the key K. The key becomes K^+. opad and ipad are the internal and key-related bit sequences of HMAC. The HMAC algorithm structure diagram is shown in Fig. 1.

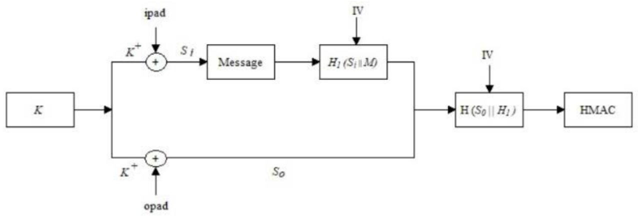

FIGURE 1. HMAC algorithm structure diagram.

The encryption process for each message block is divided into five steps [18]:

Firstly, make the number of bits of the key K the same as the number of bits b in each Hash function grouping by adding zeros to the last bit to obtain K^+.

Secondly, performs an equal or operation on the ipad to produce a grouping of b bits and appends M to it to produce a message authentication code.

Thirdly, input the message authentication code derived from step 2 into the embedded Hash function to generate the Hash code.

Fourthly, it performs an iso-or operation with opad to generate a grouping of b bits and attaches the hash code generated in step 3 to fill to the b bits, generating a new message authentication code.

Fifthly, the message authentication code generated in step 4 is directly applied to the Hash function to generate an HMAC value.

The HMAC value generated after the encryption of the previous message block is used as the initial value for the subsequent message block processing, and so on repeatedly until the last message block processing is completed to get the final pseudo-random number output value.

B. CHMAC ALGORITHM

According to the working requirements of the RTPP system, the HMAC algorithm should be improved in terms of time and energy consumption. Therefore, we tried to find a way to optimize the time consumption of private data encryption in HMAC. For this purpose, we conducted a series of tests and evaluations to find the most time-consuming part of the HMAC algorithm as a possible option to improve the algorithm running time. Each round of the HMAC algorithm contains three calls to the hash function, the main core of the algorithm, which processes messages in 512 bits increments, with the internal structure of each round consisting of permutations, shifts, and substitutions. Contrary to the simple and low-cost implementation of bit permutations in hardware [19], the software implementation is expensive from the aspect of processing time. Therefore, to further improve the security and encryption speed of the HMAC algorithm, this paper proposed the embedded Hash function in the HMAC algorithm. Combining the segmented logic chaos mapping with the embedded Hash function and invoking the Piecewise Logic Maps (PLM [20]) to construct the CHMAC algorithm. Our experience reduces the processing time of the software by reducing the number of substitution operations, while ensuring better encryption performance. Table 1 shows the time (milliseconds) required to encrypt 512 bits of data with different encryption rounds for the HMAC algorithm, the improved HMAC algorithm and the CHMAC algorithm proposed in this paper. The results show that the encryption time for encrypting 512 bits is reduced from 255.07ms to 20.97ms when the HMAC algorithm does not include the permutation operation. The CHMAC algorithm retains one permutation and introduces chaotic mapping. From comparing of encryption iteration times, the CHMAC algorithm has an encryption

TABLE 1. Require time for encryption data (512 bits) in different scenarios with the different number of rounds (millisecond).

Encryption Algorithms	2	4	6	8	10	12
HMAC algorithm	42.81	85.11	127.72	170.1	212.47	255.07
HMAC algorithm-without permutation	4.63	7.91	10.97	14.38	17.63	20.97
HMAC algorithm-with one permutation	24.98	27.84	30.26	33.79	36.94	40.08
CHMAC algorithm (proposed)	25.46	28.51	31.69	34.95	38.07	41.16

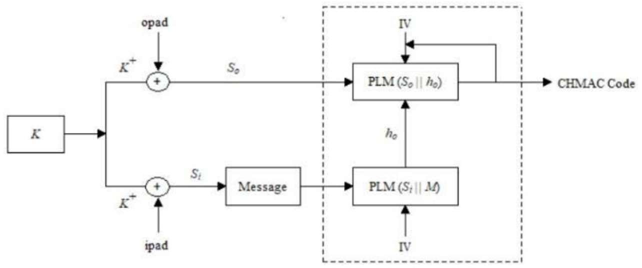

FIGURE 2. CHMAC algorithm structure diagram.

advantage over the HMAC algorithm which has only one swap.

In CHMAC algorithm, the introduction of chaotic mapping reduces the interaction between plaintext information blocks in the initial stage, effectively prevents external attacks, and greatly improves the algorithm's security. The logistic map is a discrete-time dynamic system [20], being mathematically expressed as

$$x_{n+1} = f(x_n) = \mu x_n(1 - x_n) \tag{2}$$

where $x_0 \in (0, 1)$ is the state value, and μ is the control parameter. The basic logistic mapping is vulnerable to attacks due to its simple structure. this algorithm references PLM [20], enhances the resistance of logistic mappings, which is defined as (3). Where N is the number of segments of the logistic mapping. It has good ergodicity and a larger Lyapunov exponent than basic logistic mappings. The study shows that the mapping has good chaotic characteristics when the initial control parameter values $\mu \in (2, 4)$.

Fig. 2 depicts the structure diagram of the CHMAC algorithm constructed in this paper. Compared with the HMAC structure diagram, the encryption of each group of messages only needs to be run twice in the same Hash function. It simplifies the design of the circuit while ensuring the improved security of the encryptor. The specific encryption process is as follows: firstly, the key and are subjected to the iso-or operation, and the generated message authentication code is input to the CHMAC algorithm structure to generate the Hash code; then, it is input to the PLM($S_o||h_o$) in the CHMAC algorithm structure to complete the second encryption operation, and the CHMAC code of a single message block can be obtained after the completion of the iteration.

$$x_{j+1} = \mathrm{PLM}\left(x_j\right) = \begin{cases} N^2\mu x_j\left(\frac{1}{N} - x_j\right), & 0 < x_j < \frac{1}{N} \\ 1 - N^2\mu\left(x_j - \frac{1}{N}\right)\left(\frac{2}{N} - x_j\right), & \frac{1}{N} < x_j < \frac{2}{N} \\ \vdots & \\ N^2\mu\left(x_j - \frac{i-1}{N}\right)\left(\frac{i}{N} - x_j\right), & \frac{i-1}{N} < x_j < \frac{i}{N} \\ 1 - N^2\mu\left(x_j - \frac{i}{N}\right)\left(\frac{i+1}{N} - x_j\right), & \frac{i}{N} < x_j < \frac{i+1}{N} \\ N^2\mu\left(x_j - \frac{N-2}{N}\right)\left(\frac{N-1}{N} - x_j\right), & \frac{N-2}{N} < x_j < \frac{N-1}{N} \\ 1 - N^2\mu\left(x_j - \frac{N-1}{N}\right)\left(1 - x_j\right), & \frac{N-1}{N} < x_j < 1 \\ x_j + \frac{1}{100N}, & x_j = 0, \frac{1}{N}, \frac{2}{N}, \ldots, \frac{N-1}{N} \\ x_j - \frac{1}{100N}, & x_j = 1 \end{cases} \tag{3}$$

Finally, the CHMAC code is fed back to the initial value of the function, and the above steps are repeated until all message block groupings have all executed this process, and pseudo-random number encryption of privacy can be realized. The processing of the function part consists of three main steps: message key preprocessing, compression iteration of the message block, and generation of the CHMAC value. Equation 4 defines the CHMAC algorithm:

$$\mathrm{CHMAC}_K(M) = \mathrm{PLM}[P_0||S_i] \tag{4}$$

Message key preprocessing consists of two parts: message key padding and message code iterative chunking. First, the key K^+ and ipad perform the iso-or operation to divide the plaintext message into L groups of plaintext message blocks Y_i $(0 \leq i \leq (L - 1)$, and after merging the two, they form the message key S_i. The length of each message key is 512 bits

TABLE 2. **Chaos-based HMAC algorithm.**

```
Let   $x_0 = S_i$ , $\mu = 4$ , N = 64 , n = 512
For   $j = 1$ to n
  $h_j \leftarrow \mathrm{PLM}(S_j)$
  $x_j \leftarrow \mathrm{PLM}[(h_j + x_{j-1})/2]$
Int  $i = 1$ to n
IF   $x_j > 0$ && $x_j \leq 0.5$
   then  $H_i = 1$
END IF
IF  $x_j > 0.5$  && $x_j \leq 1$
   then  $H_i = 0$
      END IF
```

and sent to the function for iterative compression, and finally, get the 256 bits code (h_o). Then use it as the expansion bit generated by the key and for the iso-or operation. Then enter the function again for iterative compression. The CHMAC code value of this message block can be generated and used as the initial value for the next group of message blocks to be processed until all the message blocks of the message are processed. Then the final CHMAC code value can be obtained. This enhances the diffusion effect among message blocks and enhances the security of encrypted messages. The iterative compression process of message blocks is mainly used in the PLM iterative function. Table 2 shows the execution process of the CHMAC algorithm.

IV. APPROACH TO "AUTO-FORGOTTEN" IMPLEMENTATION FOR EMBEDDED SYSTEM

One way to protect private data in storage and achieve an automatic deletion to implement the "right to be forgotten" is to limit users' private information [21]. However, as data grows, the number of files that need to be deleted gradually increases the complexity of system processing. So far, the deletion operations users have performed on the device have only ostensibly been deleted on their own devices. The data system backend has saved this information in the backend database of each company [22]. Whenever a company receives the requirement to erase personal data from the database, the whole process of individual-by-individual review is very tedious and time-consuming. However, the final review decision does not always ensure successful deletion, reducing the legal system's credibility to protect individuals' privacy.

Protecting individuals' privacy through legal means is not a foolproof solution, so it is crucial to deal with the "automatic right to be forgotten." In this paper, a Resilient Tag-based Privacy Protection (RTPP) scheme is designed to solve such problems effectively. The scheme automatically calculates the survival period of individuals' private data by controlling the charge movement in the hardware and changing the bit-error rate (BER) in combination with the data usage in a specified period. When the data is outside the survival cycle, it will be automatically and permanently destroyed in the hardware to be forgotten. Fig. 3 is the basic architecture of

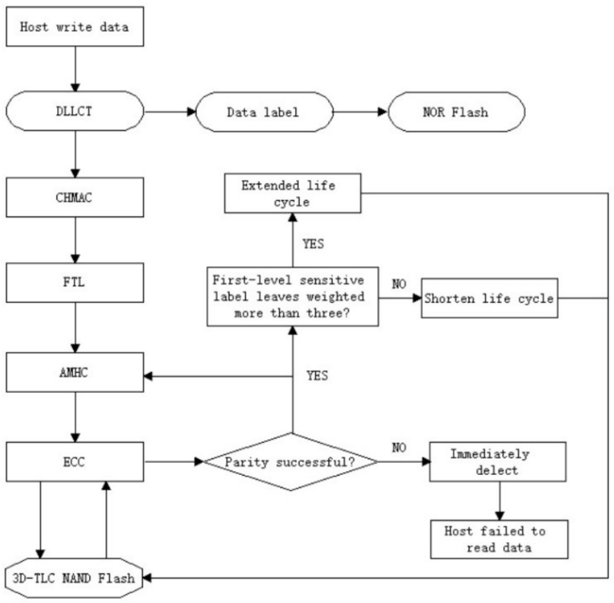

FIGURE 3. **Architecture of the RTPP system.**

the RTPP scheme. The main features of the RTPP system are: firstly, private data all have corresponding tags; secondly, the existence time of individuals' privacy can be flexibly adjusted by determining whether the user retrieves the relevant tag data for four out of seven days; thirdly, all private data under such tags can be accurately operated by directly retrieving the tags without decoding the private data. This system consists of four parts: first of all, using a pseudo-random number generator with chaotic mapping to perform cryptographic operations on privacy; next, using NOR flash memory and 3D-NAND flash memory controllers for collaborative processing; then controlling the length of personal privacy lifecycle with Huffman coding; finally achieving automatic forgetting of individuals' privacy. The features are described in detail in Section 3 of this paper.

A. *FLASH MEMORY OPERATION*

There are two typical types of flash memory, NAND and NOR flash memory [23]. However, NAND flash memory is classified into four types based on the difference in density of its electronic cells. After comparing the capacity, cost, and lifetime of these four types of flash memory, 3D-TLC NAND flash memory [24] was chosen for this system. 3D-TLC NAND flash memory is not simply a stack of NAND layers. It utilizes 3D-NAND technology, where memory particles are stacked in three dimensions from three dimensions. It dramatically improves storage capacity, performance, and security compared to two-dimensional planar-sized TLC NAND flash and has an advantage over two-dimensional flash in storing large-capacity private data [25]. NOR flash is a random storage medium. Each memory cell is connected in parallel, allowing direct random access to each bit and significantly reducing the execution time for processing instruction operations to store data tags in NOR flash [26]. When the host issues a retrieval command, it first extracts the relevant tag from the NOR flash memory and sends it to the NAND flash memory. Then, it can view the data corresponding to this tag and transfer the data to the host to complete this retrieval operation. Similarly, when a host wants to delete a particular type of data, it can directly delete such tags and delete all the data under such tags simultaneously to achieve flexible regulation of the data lifecycle. Fig.3 designed the RTPP system to use two flash memory types for individuals' privacy.

Although the storage performance of the two types of flash memory is very different, the read and write processes are similar [27]. For NAND flash, the deletion or writing of data is based on the tunneling effect, which requires current to pass through the insulation layer between the floating gate and the polysilicon pillar, discharging or charging the floating gate [28]. NOR flash memory uses tunneling for data deletion and hot electron injection from the floating gate to the source for data writing [29]. In order to achieve flexible control of the survival cycle of individuals' privacy, the proposed RTPP system designed in this paper utilizes the charge movement to control the erasure and writing of flash memory. The tags with private data are stored in 3D-TLC NAND flash memory, and each layer stores one week of private data. When the data life cycle is extended, the content of the bottom tag is substituted to the tag with the same name in the upper layer. By controlling the charging and discharging of the bottom cell, the outdated tag is erased while the data is written. At this time, the error correction code will receive the corresponding instruction to determine whether the BER should be increased or decreased [30], thus realizing the automatic adjustment of the private data life cycle by flash memory. The operation does not require direct private data processing but compresses and stores them in their respective tags. Our system only needs to manipulate the corresponding tags to achieve control over the life cycle of all private data, which saves memory processing time, dramatically improves efficiency, and effectively protects the privacy and security of users.

B. RULES OF LABELING DATA TAGS

After the server is written with individual private data, it first classifies each private data by labeling it with a corresponding tag and stored in the flash memory. In the DLLCT designed in this paper, each tag type has its corresponding timeline from creation to disappearance. Its lifecycle is automatically updated in the table when the private data life span needs to be extended, shortened, or deleted immediately. The tags in the life cycle table are divided into four groups, among which the first three groups of tags are fixed in position and value in the life cycle table and cannot be modified in any way. At the same time, the system automatically generates the fourth group of tags according to the sensitive level of privacy.

All tags are stored in the NOR flash memory of the embedded system as a server host. When users use the host, the generated privacy content will look for the tags matching their own inside the host to realize the categorization and storage of private information. At this time, each private data can be labeled by multiple tags, and different types of sub-tags can be stored under each group of tags. Each tag is stored in the life cycle table with a default validity of one year. If no operation is performed on these private data during this period, this private data under such tag will automatically be destroyed in the system. If the data is subject to an extended period, shortened period, or immediate deletion operation, the survival time of its corresponding life cycle table will also be automatically changed.

On the one hand, the tag is stored in NOR flash memory so that the flash memory can directly handle a large amount of private information. On the other hand, the tag and the private data it contains are transferred to a pseudo-random number generator based on the chaos principle, which encrypts the data information to prevent private data leakage. The use of pseudo-random number generator based on chaos principle and its encryption principle is described in detail in Chapter 2. When the private data has completed the above operations, it will enter the embedded system's Flash Translation Layer (FTL) [31]. This step converts the logical address of the private data into a physical address for writing to flash memory. After the conversion is completed, the privacy information is directly input into the Huffman coding designed in this paper to compress the private data. The regulation of the Huffman encoding is the core part of completing the automatic regulation of the private data, which is explained in detail below.

C. MODIFICATION OF HUFFMAN CODING

After the data are tagged, the random encryption of the pseudo-random function generator is initialized, and FTL completes the address conversion. It enters the core module of the RTPP system, which is a crucial step used to realize the flexible regulation of the life cycle length of private data. This paper designs an algorithmic modulation of Huffman coding to achieve lossless compression of large amounts of private data using Huffman coding. The system flexibly changes the life cycle of private data in flash memory by judging the weight results of the Huffman tree so that the error correction code generates the corresponding bit error rate and thus controls the directional movement of the flash memory charge.

Huffman coding algorithms have two manifestations in the current research: static mode [32] and adaptive mode [33]. Throughout the encoding process, the static encoding model bases the encoding process on a pre-assumed model of the distribution of encoded elements and allows the use of character distributions that correspond to the nature of the file. Our auto-adaptive algorithm does not lose compression gain if the differences between the presumed and actual models are too significant because it draws on the model details of the incremental model. When there are significant changes in the patterns of different elements, the adaptive approach also does not need to transfer these changes to the decoder.

Because in this mode, the encoder and decoder automatically keep the identical copy with the Huffman tree, thus showing that the adaptive mode is better than the static mode is more advantageous than the static mode. Although the RTPP system already provides a life cycle table of data tags, some of the more petite tags in group 4 can only be written to the life cycle table by the user. Thus, to make Huffman coding more effective in regulating the life cycle of private data in RTPP systems, this paper designs an Adaptive Model of Huffman Coding (AMHC).

For Huffman coding, the construction of Huffman tree is the most fundamental work. In this paper, we design an adaptive dynamic mode of Huffman coding. The Huffman tree is based on the tag information in DLLCT as the basic structure, and in the actual use, the user's Huffman tree is constructed step by step backward according to the date of each day, and the whole tree is not completed at the beginning. The process of its construction is rough, using the first set of tags as the root node and building the leaf nodes sequentially from top to bottom. The second group of month tags in the life cycle table is read as the leaf node of the root node, where the current month tag is placed in the left node. The right node is the next month tag; the third group of week tags is used as the leaf node of the second group of tags, with the current week tag as the left node and the right node as the next week tag. For the construction process of the leaf nodes of the second and third groups of tags, the above method is repeated in turn until the last tag in the second and third groups of tags in the life cycle table appears. It completes the construction process of the first three groups of tags for the whole year Huffman tree. The leaf nodes of the third group of tags are constructed according to the fourth group of user tags. The leaf nodes are sorted according to the order of user accesses built in order from left to right. Since the fourth group of tags is classified by the user's private data sensitivity to the server, the initial weight of the leaf of the data with the highest sensitivity is set to 1. The weight is set to 2 to a higher sensitivity level, and so on. The Huffman tree of the fourth group tags is constructed with weights after tags are classified. This Huffman tree is merged under the third group of leaf tags to complete the construction of the Huffman tree of a user's private data in a day in the server. The leaf weight of a user's first-level sensitive tag is consistent with the number of days a user visits the server. If a user visits the server four days a week, its first-level sensitive tag leaf weight changes to 4, and the weights of all the remaining leaf nodes change accordingly. If a user visits the server frequently, the amount of his privacy record data increases, increasing the risk of privacy leakage. The server has specific protection measures for their private data for this type of user.

Since all groups of tags are set to be valid for one year by default, Huffman coding sets a timeline every seven days. By determining whether the leaf weight of the first level-sensitive tag of the fourth group of tags is greater than 3, it is possible to decide whether the life span of the fourth group of tags is extended or shortened. When the determination is over, the weights of all tags in the fourth group of that user change to the initial weights, and then the task of regulating the data life cycle is performed. If it is larger than three, the Huffman tree changes at that time: the first three levels of sensitive tags in the fourth group of tags are then set to shorten the life cycle, and the server sets its initial leaf weight to decrease by one-twelfth, and the remaining sensitive tags of this user are set to extend the life cycle, and their initial leaf weights increase by one-twelfth; if it is less than or equal to three, all the fourth group of tags of this user is set to shorten the life cycle, and its leaf initial weight is reduced by one-twelfth. For a user with an extended lifecycle, when the leaf weight of the first three levels of sensitive tags is reduced to 0 within the one-year validity period, the user will not display the contents of the first three levels of sensitive tags when he/she revisits the server. At that time, the user's fourth level of sensitive tags becomes the new first level of sensitive tags, and its leaf weight becomes 1. The fifth level of sensitive tags becomes the new second level of sensitive tags, and its leaf weight becomes 2. If the user revisits the server, his privacy tag will not participate in the construction of the Huffman tree, and the system will directly include this user in the critical protection list. When the one-year validity period expires, the system will directly set all tag leaf weights to 0. At this time, it enters the automatic forgotten phase of the embedded system. Until the second year, the above process starts again. Table 3 shows the implementation process of the AMHC algorithm. When the system receives the instruction to extend the tag life cycle, it reduces electrons' migration and error rate to the 3D-TLC NAND flash interface. Thus, the error correction code does not easily reach saturation, and the data lifecycle extension is achieved. Instead, it will increase the charge migration on the 3D-TLC NAND flash interface and increase the error rate, allowing the error correction code to detect more errors, thus shortening the data lifecycle. For leaf node tags with a weight of 0, the system will remove them before entering the second cycle.

Suppose a user sends a request to delete private data immediately while using the host system. In this case, the system first finds which type of tag the private data belongs to in the periodic table. Our system retrieves its usage frequency in the Huffman tree by the fourth group of tags and immediately reduces its leaf node weight to 0. In the Huffman pseudo-code, this leaf node is simultaneously deleted in the Huffman tree, and its weight in the periodic life table will be deleted accordingly. At this time, the error correction code reaches the maximum error correction value, the parity check fails, and the user-submitted privacy deletion instruction enters the hardware immediate deletion phase. A large amount of charge will be transferred, and permanent hardware deletion of this private data is finally achieved after the discharge operation [34]. Since the private data is stored under tags, in this case, the deletion operation is performed directly on all the tags owned by this private data to achieve the deletion of private data.

TABLE 3. AMHC algorithm.

```
Int   $i, j, t$
Let   n = 365       Data Life Cycle
Initialize a Huffman tree according to the sorting of tags in the
For  $i$ = 1 to n
          $x_i \leftarrow$ the fourth group of i − level − sensitive leaf tag
output the Huffman tree for the fourth set of tags
For  $j$ = 2 to n
    IF  $x_1$ is repeated
        $x_1$'s weight plus one
        IF $j$ is a multiple of 7
            IF  $x_1$'s weight is greater than three
                For  $i$ = 1 to $t$
                    $x_i$'s weight becomes the initial value
                $x_1$'s, $x_2$'s, $x_3$'s weight is reduced by one
               − twelfth
                $x_i$'s weight is increased by one − twelfth
      ELSE
                For  $i$ = 1 to $t$
                    $x_i$'s weight becomes the initial value
                $x_i$'s weight is increased by one − twelfth
            IF $x_i$'s weight is 0
                delete the leaf tag $x_i$
```

D. DLLCT WORKING PROCESS

DLLCT is a key step in the RTPP scheme to achieve flexible extension/shortening of data lifecycle. First, it judges the Huffman tree's leaf weights to change the private tag's lifecycle. It then sends the corresponding instructions to the 3D flash memory to change the BER of the patient tag in this embedded system by controlling the direction of the electron flow at the flash interface to complete the change of the private data lifecycle, Fig. 4 shows the way of working of DLLCT in RTPP scheme. The figure shows that DLLCT first sets all the private data tags that enter the system after encryption to be valid for 1 year. At the same time, the AMHC algorithm starts to work, at which time the processing of data tags enters the working mode ② and ③, during which time if the system receives the command to delete the data immediately, it will enter the working mode ④ at this time. Fig. 5 depicts the workflow diagram of DLLCT for flexible regulation of private data life cycle. Firstly, DLLCT will estimate the BER based on the private data and optimize the leaf labels' weights within seven days. The actual BER will be calculated on the eighth day, and the flexible control of the private data life cycle can be realized.

V. SYSTEM IMPLEMENTATION RESULTS ANALYSIS

In the proposed RTPP system, there are two core components, one is the encryptor, and the other is the AMHC implementation. In the following, we will analyze the RTPP system from two aspects: the security analysis of the system, and the process performance of automatic adjustment of the private data lifecycle by AMHC.

A. SECURITY ANALYSIS OF THE SYSTEM

The security of the RTPP system is mainly reflected in the system's resistance to attacks and the security of storing

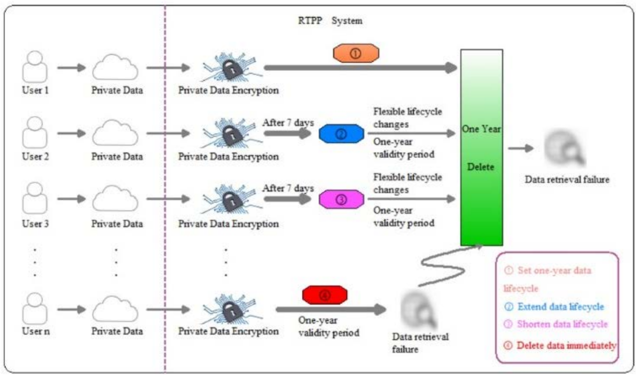

FIGURE 4. Proposed Data Label Life Cycle Table (DLLCT).

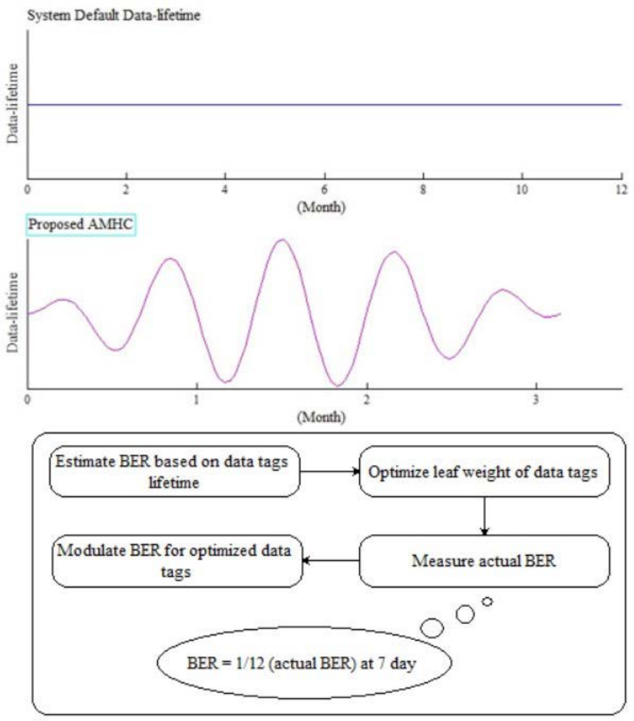
System Default Data-lifetime
Data-lifetime
(Month)
Proposed AMHC
Estimate BER based on data tags lifetime
Optimize leaf weight of data tags
Measure actual BER
Modulate BER for optimized data tags
BER = 1/12 (actual BER) at 7 day

FIGURE 5. Flowchart of proposed DLLCT.

private data in the system. The performance index of the encryptor in the RTPP system can be tested, and the security analysis of the password can be judged. We tested the proposed CHMAC algorithm in the RTPP system and compared it with PRNG algorithm based on the Hash function and the PRNG algorithm based on the MAC function. We compared the three algorithms in terms of energy consumption, encryption time, and memory usage to evaluate the overall performance of the CHMAC algorithm. Evaluate the system's resistance to attacks by studying the relationship between plaintexts and keys generated by the CHMAC algorithm. The following six experimental results show that the CHMAC algorithm introduces chaotic mapping and uses nonlinear elements compared with the Hash function-based PRNG algorithm and the MAC function-based PRNG algorithm.

Although its performance index is between the two, its resistance to attacks is the strongest and provides a stronger security defense for the RTPP system.

1) ENERGY CONSUMPTION OF ENCRYPTION

For electronic devices, the battery is the direct component that provides energy, so we will calculate the energy consumption of the encryptor by measuring the usage of the battery by the encryption algorithm. Using a multimeter to measure the voltage and current values required for the algorithm to run, we will first find the power when the algorithm runs, according to the formula: power = voltage value * current value, $P(w) = U(v) * I(A)$, the power value is obtained. In this formula, the voltage and current values are taken as the average of the measurement results of the algorithm run thirty times. The average power is obtained and brought to the formula: $Q(J) = P(w) * T(s)$, which gives the amount of energy consumed by each encryption algorithm to run.

In this case, T is the time required to execute the algorithm once, and its value remains the average time of thirty measurements. Fig. 6 shows the energy consumption required by the three encryption algorithms to execute 128 bytes, 256 bytes, and 512 bytes of data. From the figure, it can be seen that the energy demand for running the CHMAC algorithm lies between the two. Since energy consumption is directly related to the algorithm's complexity, one cannot judge whether an encryption algorithm is good or not only by the degree of energy loss. Among the three algorithms, the CHMAC algorithm introduces chaotic mapping into the embedded Hash, increasing the algorithm's complexity and improving encryption security.

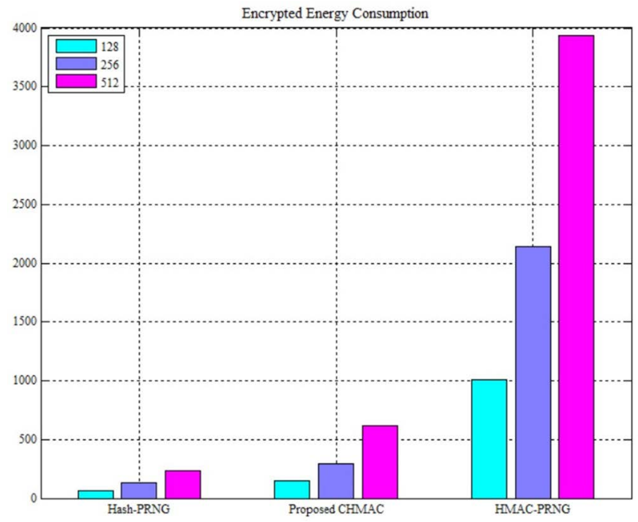

FIGURE 6. Average energy consumption for three encryption algorithms executing three bytes (MJ).

2) TIME CONSUMPTION OF ENCRYPTION

In addition to energy consumption, the algorithm's execution time is also a critical factor in determining its performance. In general, the higher the algorithm's complexity, the faster it completes encryption and the better the algorithm's performance. The time consumed by the three algorithms to execute 128 bytes, 256 bytes, and 512 bytes of data is shown in Fig. 7. The Hash algorithm has the fastest completion time because it has the lowest complexity. However, the security of the keys it generates is less than that of the other two algorithms. Therefore, although the Hash algorithm has the fastest execution speed, it is not the best algorithm. Compared with the HMAC algorithm, the CHMAC algorithm improves the process of message grouping iterations, shortening the execution time of data encryption.

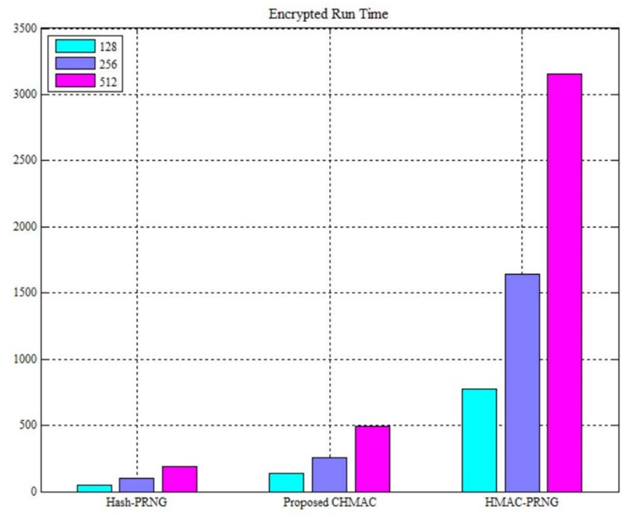

FIGURE 7. Running time for three encryption algorithms executing three bytes (ms).

3) THE MEMORY OCCUPATION OF ENCRYPTION

The proposed RTPP system works in standby mode when the host generates browsing data. Once new private data is added, RTPP will immediately enter working mode. Therefore, the system memory needs to be occupied only with encrypting the private data or adjusting its life cycle. The adjustment data lifecycle phase mainly uses NOR flash memory and 3D-NAND flash memory, which requires more system memory in the encryption phase. Due to the limited memory, it is essential not to occupy too much memory while ensuring the encryption speed and quality. Therefore, the amount of memory required to run the encryption algorithm is generally considered memory RAM usage [35]. Fig. 8 Shows the RAM usage of the three encryption algorithms, and it can be seen from the figure that the Hash algorithm has minor RAM usage, followed by the CHMAC algorithm and the HMAC algorithm. Although the memory usage of CHMAC algorithm is not the least, the process of CHMAC algorithm encryption is the most complicated among these three algorithms. In a comprehensive view, CHMAC algorithm is still the best.

The above three aspects of the evaluation results prove that the proposed CHMAC algorithm designed in this

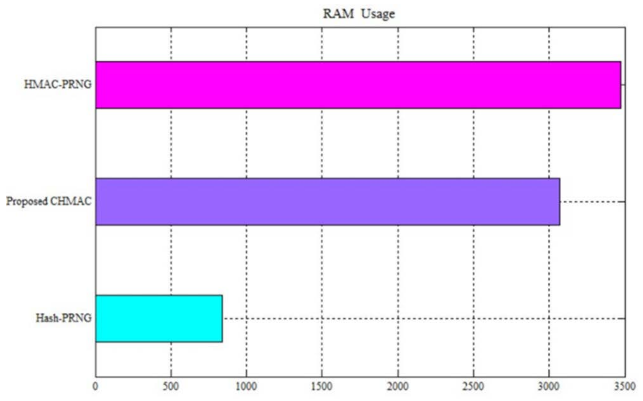

FIGURE 8. Memory usage of the three encryption algorithms (Bytes).

paper is slightly better performance. However, none of them is the smallest in terms of algorithm complexity, the anti-interference ability of encryption.

For encryption algorithms, the length of the initial key determines the security level of the encryptor; the longer the key, the more resistant the encryption algorithm is to attack and the higher its security. In terms of performance, the longer the key, the longer the process of compressing and iterating the key by the encryptor, the longer the encryption time consumed by its complete encryption process, and the more RAM it takes up. The CHMAC algorithm achieves the optimal security of the encryptor without increasing the performance cost. In the following, three aspects of the correlation between the ciphertext and plaintext generated by the CHMAC algorithm, randomness, and resistance to attack will be analyzed. In order to make the analysis results more accurate and reliable, 200 random text samples were generated by the Lorem-Ipsum library to participate in this experiment [36]. Among them, 100 random texts have a size of 5000 bytes, and another 100 random texts have a size of 10000 bytes.

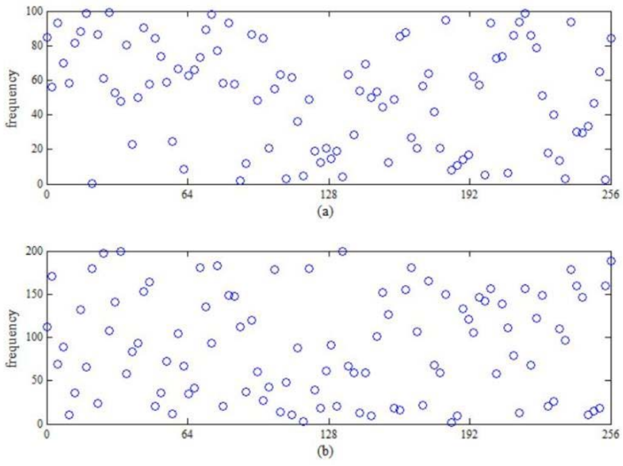

FIGURE 9. ASCII distribution of plaintext characters for 200 random texts in the CHMAC algorithm.

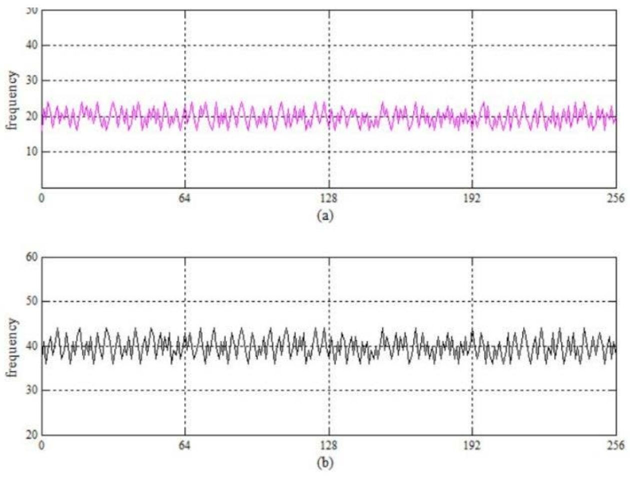

FIGURE 10. ASCII distribution of ciphertext characters for 200 random texts in CHMAC algorithm.

4) CORRELATION OF PLAINTEXT AND CIPHERTEXT

For an encrypted ciphertext, the less correlation it has with the plaintext, the less the attacker can get the related plaintext content, and at this time, the more secure the plaintext is, the less the private content can be revealed. The correlation between plaintext and ciphertext can be determined by counting the ASCII characters values in the plaintext and the ciphertext. As long as the ASCII value distribution of the plaintext and the ciphertext does not show any pattern, it proves that the plaintext and the ciphertext are not correlated, and the private information after encryption is secure. For the formed ciphertext, 0 to 256 ASCII characters indicate that the encryption is secure and the formed ciphertext has low predictability. Fig. 9 (a) and (b). show the ASCII distribution of characters in plaintexts of 5000 and 10000 bytes, respectively; Fig. 10 (a) and (b). are the ASCII distributions of characters in the ciphertext after encryption for two random texts without size bytes. Comparing the four graphs, we can see that the characters in the random text before encryption are random and irregular. After encryption, the characters are uniformly distributed, indicating that the CHMAC algorithm's encryptor has a relatively high-security index.

5) RANDOMNESS OF THE CIPHERTEXT

After the encryption process, a ciphertext containing only binary numbers is generated after the encryption process encrypts the private data. Therefore, by counting the number of binary numbers 0 and 1 generated by encryption separately, it is possible to determine whether the encryptor satisfies the characteristic of the randomness of encryption output. Theoretically, the encrypted output is best when the number of 0s and 1s is fifty percent each. At this point, the ciphertext is not easy to find the pattern, and it is not easy to be broken, which means that the generated ciphertext is secure. Table 4 shows the counts of 0s and 1s in the encrypted output after encrypting 200 random samples by the CHMAC algorithm,

TABLE 4. Average number and percentage of "0" and "1" in 200 random encrypted samples.

Random Text	0's Count	0's Proportion	Deviation from 50%	1's Count	1's Proportion	Deviation from 50%
5000 Bytes	19828	49.57%	+0.43%	20172	50.43%	-0.43%
10000 Bytes	40126	50.1575%	-0.1575%	39874	49.8425%	+0.1575%

along with the respective percentages. Here the count values of each type of bytes are obtained as the average of such texts. From the table, we can see that the average total percentage of ciphertext 0 and 1 generated by the encryptor of CHMAC algorithm is basically around 50%, and the encryptor designed in this paper fully satisfies the randomness.

The randomness of ciphertext can also be measured by information entropy, which is the discrete probability of detecting characters in a random text; the more chaotic the ciphertext is, the greater the uncertainty of each character. The information entropy value of the characters in the random text is calculated according to the formula $H(S) = \sum_{S} P(S_i)log_2\frac{1}{P(S_i)}$, where $P(S_i)$ is the probability of each ASCII occurring in the ciphertext [37]. When the value of the encrypted string is wholly distributed in the ciphertext, the salient value of information entropy is equal to 4 for a random text of 5000 bytes; the outstanding value of information entropy for a random text is that of 10000 bytes is 8. Fig. 11 is the value of information entropy for a random text. It shows that the entropy value of the ciphertext characters generated by the CHMAC algorithm is close to the entropy value of excellent information entropy, which shows that it conforms to the design principle of the encryptor.

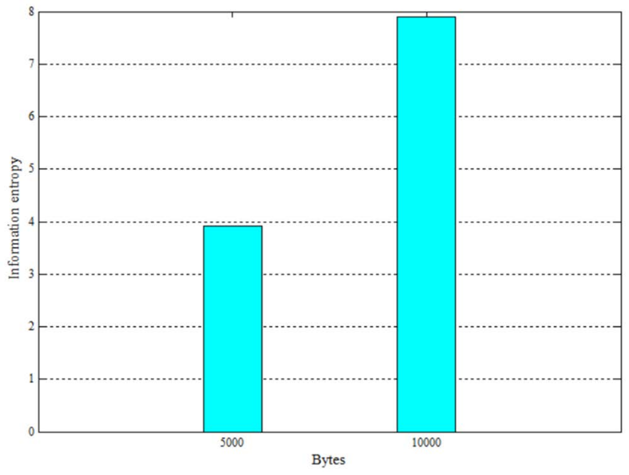

FIGURE 11. Entropy values of ciphertext information for 200 random texts in the CHMAC algorithm.

6) ATTACK RESISTANCE OF CIPHERTEXT

The ciphertext generated by a qualified cryptography must be highly resistant to external attacks to ensure that private information is secure Diffusion, obfuscation, and avalanche effect are three basic principles of cryptography design [38]. Diffusion allows each bit of information in the plaintext to affect many bits of information in the ciphertext, which can hide the contents of the ciphertext. Obfuscation makes the relationship between the statistical properties of characters between the ciphertext and the key more complex, even if the attacker obtains the relevant information of the ciphertext. The avalanche effect belongs to an unstable equilibrium state. When the plaintext or the fundamental changes slightly, the ciphertext will produce a considerable change, such as half of the binary bits in the ciphertext change in reverse. In order to test the resistance of ciphertext to attacks, the next part of this paper will analyze both the diffusion and obfuscation properties and the avalanche effect of the plaintext, key, and ciphertext parts. The diffusion and confusion properties between the plaintext and ciphertext characters in the encryption algorithm are calculated. According to the formula of integrity metric, it has a value of 1 for all encryption algorithms. According to Equation (5) [7], where n is the number of bits of the plaintext input of the encryption method and m is the number of bits of the ciphertext output generated by the encryption method.

$$d_c = 1 - \frac{1}{nm} \not\equiv \left\{(i,j)\,|a_{ij} = 0\right\}, \quad (i = 1, \ldots, n; j = 1, \ldots, m) \qquad (5)$$

Fig. 12 shows the computed results of the diffusion and confusion properties of the CHMAC algorithm. It shows that the algorithm converges to 1 after the fourth iteration, which is consistent with the diffusion and confusion properties of the cryptograph. The avalanche effect is tested here by assuming that half of the binary bits of the ciphertext will be reversed and changed when the plaintext or key changes by one bit. The avalanche effect value is calculated according to Equation (6), where '$\#X$,' 'n' and 'W_H' represent the ciphertext data count, individual data bit count, and Hamming distance, respectively. $F(x)$ is the ciphertext data, and $F(x^{(i)})$ is the ciphertext data with one difference in the i_{th} position [7].

$$d_{a_1} = \frac{1}{\#X * n}\sum_{i=1}^{n}(\sum_{x\in X} W_H(F(x) \oplus F(x^{(i)}))) \qquad (6)$$

Fig. 13 shows the change in the value of the avalanche effect for the random text. Since the number of bits of the

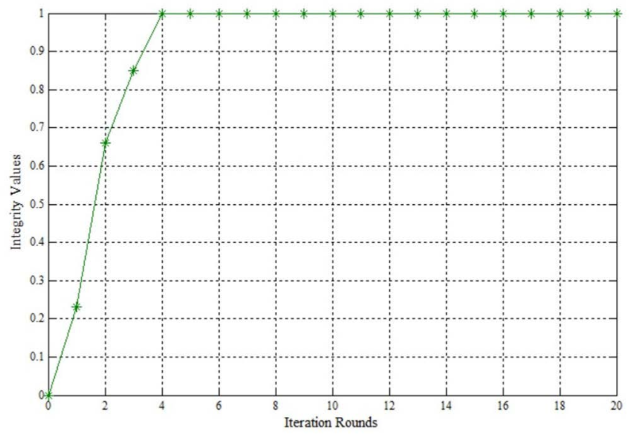

FIGURE 12. Integrity of the CHMAC algorithm.

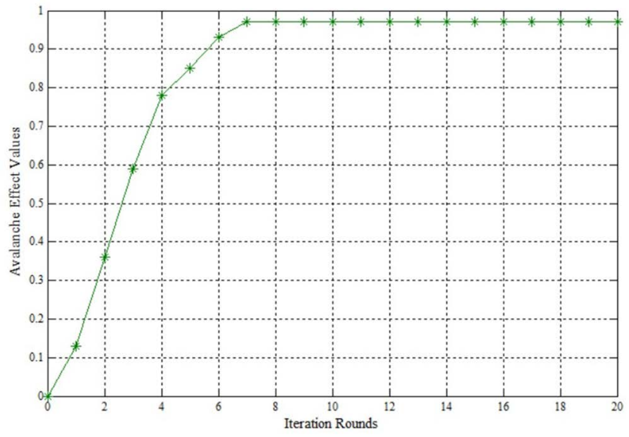

FIGURE 13. Avalanche effect of CHMAC algorithm.

ciphertext change is assumed, the outstanding value of the avalanche effect at this time should be 1. From the figure, we can see that the value of the avalanche effect of the CHMAC algorithm is closest to the ideal value after seven rounds.

B. DISCUSSION OF AMHC AUTOMATIC REGULATION RULES IN EMBEDDED SYSTEMS

The AMHC algorithm is a crucial part of the embedded system RTPP to achieve automatic data lifecycle adjustment. The AMHC algorithm achieves the compression of a large amount of private data and changes the lifecycle of private data in DLLCT by changing the weights of leaf tags. Next, we will evaluate the compression performance of the AMHC algorithm in software and the adjustment process of a data life cycle in RTPP for the embedded system. Finally, we will compare and analyze the advantages of the RTPP scheme with those of the traditional scheme.

1) COMPRESSION PERFORMANCE OF AMHC ALGORITHM

The compression performance of the AMHC algorithm is analyzed by comparing it with static Huffman coding and dynamic Huffman coding in terms of the size of the data after compression and the processing time of the compression process. Due to the complexity and diversity of private data, we selected five types of text, English, Chinese, Internet, Picture, and Random with fixed character size, for testing [35]. Table 5 is the size of the five types of text after compression by the three algorithms, where the second column is the initial size of the five types of text (in MB), and the third column is the size after compression by the three algorithms (in Byte). It was evident from the figure that the size of the data compressed by the AMHC algorithm is smaller than the other two algorithms. Some of the data compressed by dynamic Huffman coding is smaller than static Huffman coding. In addition to the size of the compressed data, the compression time is also a critical factor in excellent compression performance. Table 6 shows the compression times for the five types of text under the three compression methods, and this time is the average time obtained for each type of text executed 100 times in each algorithm. As can be seen from the figure, static Huffman coding is the fastest, dynamic Huffman coding is about half the time used for static Huffman coding, and the AMHC algorithm takes a little bit slower than dynamic Huffman coding. Although the execution time of the AMHC algorithm is slightly longer, the compression performance of the AMHC algorithm should be considered better among the three in terms of the functions it implements and the size of the compressed data.

2) PERFORMANCE OF DATA LIFECYCLE ADJUSTING IN RTPP SCHEME

Each tag is valid for one year in DLLCT, and it is up to the AMHC algorithm to decide whether to extend or shorten the life cycle of the tag. The key to this algorithm is the construction of the algorithm tree. The weights of the leaf tags of this tree will change every day. By determining whether the leaf weight of the first level-sensitive tag of the fourth group of tags is greater than 3, the system determines whether the life cycle of the user's fourth group of privacy tags is longer or shorter. Suppose the leaf weight of the first-level sensitive tag is greater than 3. In that case, the first three levels of sensitive tags of the fourth group of tags are set to shorten the lifecycle with a one-twelfth decrease in leaf weight, and the remaining sensitive tags are set to extend the lifecycle with a one-twelfth increase in leaf weight. The related tags in the DLLCT will be recorded one by one to extend or shorten the lifecycle. If the leaf weight of a level 1 sensitive tag is less than 3 and no command is issued to delete the data immediately. In this case, the life cycle of that data is automatically shortened at this point by default. Then all of its fourth set of sensitive tags are recorded as shortened lifecycle, and the leaf weight is reduced by one-twelfth. When extending the data lifecycle, a small number of electrons will flow at the 3D flash interface, reducing the BER and achieving an extended data life cycle. Fig. 14 clearly shows that the electron influx at the flash interface dominates, the number of electrons at the oxide interface increases, and the

TABLE 5. Compression performance of three algorithms.

File	INITIAL SIZE (MB)	Static Huffman coding (Byte)	Dynamic Huffman coding (Byte)	Proposed AMHC (Byte)
English	50	29846417	29846208	29846091
Chinese	50	35861703	35861518	35861106
Internet	6.8	1459762	1601870	1441951
Picture	3.5	583496	605527	582401
Random	4.6	1276359	1317287	1255360

TABLE 6. Coding execution time of three algorithms.

File	INITIAL SIZE (MB)	Static Huffman coding (Byte)	Dynamic Huffman coding (Byte)	Proposed AMHC (Byte)
English	50	0.094	0.198	0.214
Chinese	50	0.106	0.227	0.235
Internet	6.8	0.078	0.169	0.174
Picture	3.5	0.062	0.135	0.158
Random	4.6	0.071	0.149	0.169

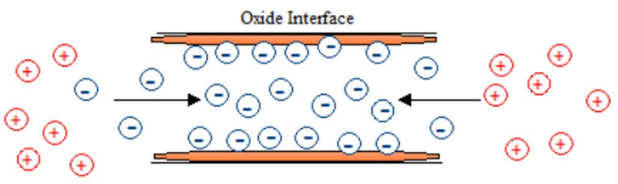

FIGURE 14. Electronic migration for BER reduction.

BER of the data changes from week to week. Fig. 15 shows the change in BER over eight days for the extended data life cycle. The BER decreases by one-twelfth for an extended data life cycle. When shortening the data lifecycle, there will be a little electron outflow at the 3D flash interface so that the BER will increase by one-twelfth consequently. Fig. 16 shows the predominance of outflowing electrons and the decrease of electrons at the interface of the oxide layer. Fig. 17 shows the change of BER within eight days for shortening the data life cycle.

3) PERFORMANCE OF IMMEDIATE DATA DELETION IN RTPP SCHEME

When the system receives a delete command for data, the AMHC algorithm will immediately zero the weight of the leaf tag corresponding to this data, and the related tags in DLLCT will be deleted accordingly. At this time, many electrons will flow out at the 3D flash interface so that the BER reaches the maximum, and the immediate delete command of the data is realized. Fig. 18 is a schematic of the electron flow, which shows that almost no electrons are present at the oxide layer interface. Fig. 19 is a hypothetical. The immediate deletion command is received on the fourth day, and the BER of this data changes in eight days.

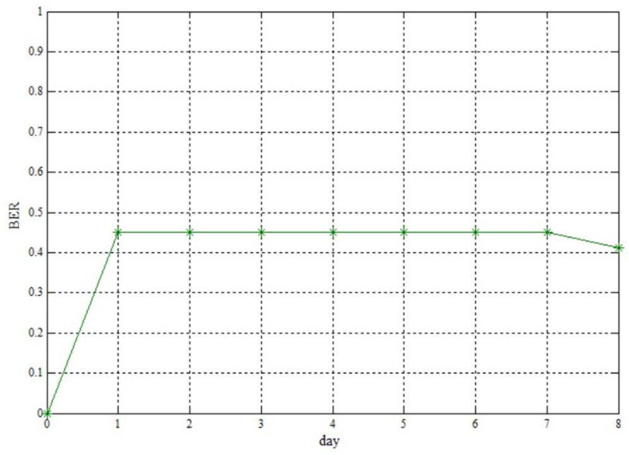

FIGURE 15. BER variation over an extended data lifecycle of eight days.

4) ADVANTAGES OF THE RTPP SCHEME AT WORK

As shown in Table 7, compared with the three traditional schemes, PP-SSS [4], Enhanced PP-SSS [5] and PDLCS [6],

TABLE 7. Comparison of RTPP scheme and traditional scheme.

system	PP-SSS [4]	Enhanced PP-SSS [5]	PDLCS [6]	proposed RTPP
Privacy data has a specific life cycle	No	No	No	Yes
Encryption is designed to encrypt private data	No	No	No	Yes
Designed a data label life cycle table	No	No	No	Yes
Flexibility to extend/shorten the data lifecycle	No	No	Yes	Yes
Delete data immediately	No	No	Yes	Yes

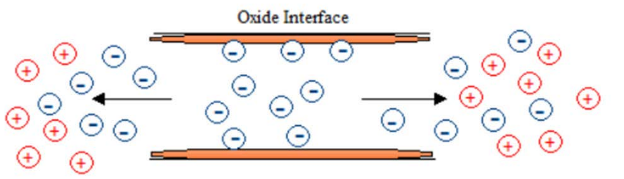

FIGURE 16. Electronic migration for increasing BER.

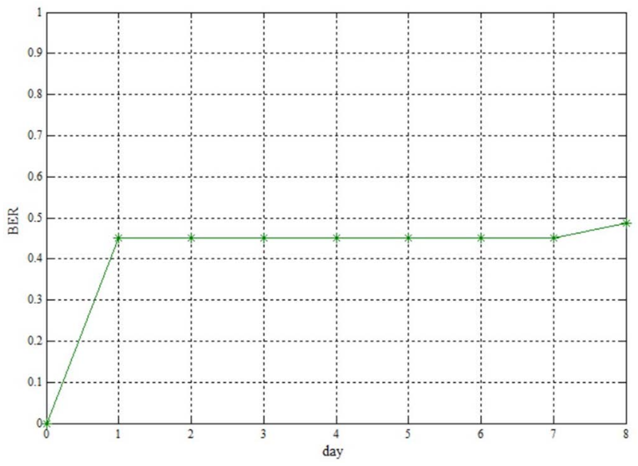

FIGURE 17. BER variation over a shortened data lifecycle of eight days.

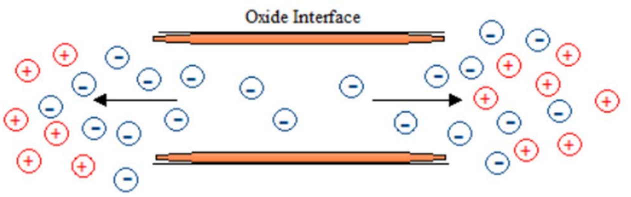

FIGURE 18. Electronic migration for maximum BER.

the RTPP scheme proposed in this paper has unique advantages in the following five aspects. The first point is that it sets a specific life cycle for private data, which saves the memory occupation of the system and effectively improves its efficiency. Traditional schemes do not have a specific lifecycle, and only change the survival cycle of data through the BER until the BER is zero before the data is permanently deleted in hardware. The second point is that the RTPP scheme is designed with encryption algorithms to encrypt private data. Only the PDLCS [6] scheme among the traditional schemes encrypts the data with a simple random encryption. In part A of this section, the encryption algorithms of the two schemes are experimented with. The experimental results show that the encryption algorithm of the RTPP scheme is significantly better in terms of performance and security. The third point is that this scheme designs a life cycle table of data label to classify the privacy data in detail, when a user's privacy is deleted, it will not affect the rest of the privacy data, and the system still works normally, the traditional scheme does not make accurate classification. The last two points compare whether this solution and the traditional solution can flexibly control the data lifecycle. The results show that this solution can flexibly extend and shorten the data lifecycle and immediately and permanently delete the data on the hardware within the specified data lifecycle. Only the PDLCS [6] scheme can do it among the traditional schemes. The core technologies of these two schemes are different; the RTPP scheme uses the AMHC algorithm and the PDLCS [6] scheme is the Inverse Huffman-Coding VTH Modulation (IHVM) algorithm, whose core ideas belong to Dynamic Huffman coding. In part B of this section, the algorithms proposed by the two schemes are compared, and the experimental results show that the algorithm of the RTPP scheme is slightly better.

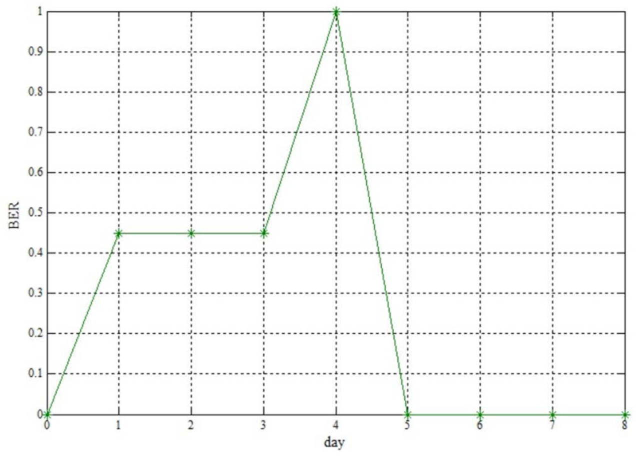

FIGURE 19. BER change over eight days for data with immediate deletion command.

VI. CONCLUSION

In order to protect personal privacy and make private data "automatically forgotten," this paper proposes a flexible and adjustable private data lifecycle control RTPP scheme for embedded systems. This system encrypts the private data using pseudo-random function cryptography based on the chaos principle and completely deletes users' private data by controlling life cycle tags. To avoid storing too much private data and occupying a large amount of system memory, the RTPP smartly links the compression of private data with its lifecycle regulation by modified Huffman coding techniques. This method can flexibly regulate the life cycle of private data, maximizing the protection of users' privacy and security issues. The proposed solution can be further improved by carrying a performance study on the security metrics in various rounds in the RTPP. It is required to probe the possibility of reducing setting groups of tags while preserving the high-security criteria and the security assessment of cryptanalytic attacks for this embedded system.

REFERENCES

[1] P. Carey, "Outsourcing personal data processing," in *Data Protection a Practical Guide to UK and EU Law*, 5nd ed. Oxford, U.K.: Oxford Univ. Press, 2018, pp. 175–176.

[2] W. Stallings, "Handling of personal information and deidentified, aggregated, and pseudonymized information under the California consumer privacy act," *IEEE Secur. Privacy*, vol. 18, no. 1, pp. 61–64, Jan. 2020.

[3] A. Bayle, M. Koscina, D. Manset, and O. Perez-Kempner, "When blockchain meets the right to be forgotten: Technology versus law in the healthcare industry," in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell. (WI)*, Dec. 2018, pp. 788–792.

[4] S. Tanakamaru, H. Yamazawa, and K. Takeuchi, "Privacy-protection solid-state storage (PP-SSS) system: Automatic lifetime management of internet-data's right to be forgotten," in *Proc. Symp. VLSI Circuits (VLSI Circuits)*, Jun. 2015, pp. C130–C131.

[5] H. Yamazawa, K. Maeda, T. Ogura Iwasaki, and K. Takeuchi, "Privacy-protection SSD with precision ECC and crush techniques for 15.5× improved data-lifetime control," in *Proc. IEEE 8th Int. Memory Workshop (IMW)*, May 2016, pp. 1–4.

[6] S. Suzuki, K. Mizoguchi, H. Watanabe, T. Nakamura, Y. Deguchi, K. Mizushina, and K. Takeuchi, "Privacy-aware data-lifetime control NAND flash system for right to be forgotten with in-3D vertical cell processing," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, Nov. 2019, pp. 231–234.

[7] Y. Liu, S. Tian, and W. Hu, "Design and statistical analysis of a new chaotic block cipher for wireless sensor networks," *Commun. Nonlinear Sci. Numer. Simul.*, vol. 17, no. 8, pp. 3267–3278, Aug. 2012.

[8] Y. Wang, K.-W. Wong, X. Liao, and T. Xiang, "A block cipher with dynamic S-boxes based on tent map," *Commun. Nonlinear Sci. Numer. Simul.*, vol. 14, no. 7, pp. 3089–3099, Jul. 2009.

[9] G. Zaibi, F. Peyrard, A. Kachouri, D. Fournier-Prunaret, and M. Samet, "Efficient and secure chaotic S-box for wireless sensor network," *Secur. Commun. Netw.*, vol. 7, no. 2, pp. 279–292, Feb. 2014.

[10] B. Liu and Q. Chen, "A method of generating pseudorandom binary sequences based on 3D chaotic mapping," in *Proc. 3rd Int. Conf. Inf. Manage. (ICIM)*, Apr. 2017, pp. 243–246.

[11] C. Jianqiu, X. Huarong, and L. Zhangli, "Image dual scrambling encryption algorithm based on parameter variable chaotic system," in *Proc. Int. Conf. Electr. Inf. Control Eng.*, Apr. 2011, pp. 4238–4242.

[12] J. Peng, S. Pang, D. Zhang, S. Jin, L. Feng, and Z. Li, "S-boxes construction based on quantum chaos and PWLCM chaotic mapping," in *Proc. IEEE 18th Int. Conf. Cognit. Informat. Cognit. Comput. (ICCI*CC)*, Jul. 2019, pp. 1–6.

[13] S. Patel, K. P. Bharath, and R. M. Kumar, "Symmetric keys image encryption and decryption using 3D chaotic maps with DNA encoding technique," *Multimedia Tools Appl.*, vol. 79, nos. 43–44, pp. 31739–31757, Nov. 2020.

[14] Y. Li and X. Li, "Chaotic hash function based on circular shifts with variable parameters," *Chaos, Solitons Fractals*, vol. 91, pp. 639–648, Oct. 2016.

[15] J. S. Teh, A. Samsudin, and A. Akhavan, "Parallel chaotic hash function based on the shuffle-exchange network," *Nonlinear Dyn.*, vol. 81, no. 3, pp. 1067–1079, Aug. 2015.

[16] S. M. S. Hussain, S. M. Farooq, and T. S. Ustun, "Analysis and implementation of message authentication code (MAC) algorithms for GOOSE message security," *IEEE Access*, vol. 7, pp. 80980–80984, 2019.

[17] Y. Yang, G. Cao, M. Qu, J. Huang, and Y. Gao, "HSATA: Improved SATA protocol with HMAC," in *Proc. 27th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2018, pp. 1–6.

[18] S. I. Naqvi and A. Akram, "Pseudo-random key generation for secure HMAC-MD5," in *Proc. IEEE 3rd Int. Conf. Commun. Softw. Netw.*, May 2011, pp. 573–577.

[19] B. J. Mohd, T. Hayajneh, and A. V. Vasilakos, "A survey on lightweight block ciphers for low-resource devices: Comparative study and open issues," *J. Netw. Comput. Appl.*, vol. 58, pp. 73–93, Dec. 2015.

[20] Y. Wang, Z. Liu, J. Ma, and H. He, "A pseudorandom number generator based on piecewise logistic map," *Nonlinear Dyn.*, vol. 83, no. 4, pp. 2373–2391, Mar. 2016.

[21] D. Erdos, "The 'right to be forgotten' beyond the EU: An analysis of wider G20 regulatory action and potential next steps," *J. Media Law*, vol. 13, no. 1, pp. 1–35, Jan. 2021.

[22] M. Nur and L. Andrawina, "Designing engineering data management system in research and development company," *J. Phys., Conf. Ser.*, vol. 1339, no. 1, Dec. 2019, Art. no. 012099.

[23] D. Zhang, H. Wang, Y. Feng, X. Zhan, J. Chen, J. Liu, and M. Liu, "Implementation of image compression by using high-precision in-memory computing scheme based on NOR flash memory," *IEEE Electron Device Lett.*, vol. 42, no. 11, pp. 1603–1606, Nov. 2021.

[24] Z. Lun, S. Liu, Y. He, Y. Hou, K. Zhao, G. Du, X. Liu, and Y. Wang, "Investigation of retention behavior for 3D charge trapping NAND flash memory by 2D self-consistent simulation," *Proc. Int. Conf. Simulation Semiconductor Processes Devices (SISPAD)*, 2014, pp. 141–144.

[25] C. Gao, M. Ye, Q. Li, C. J. Xue, Y. Zhang, L. Shi, and J. Yang, "Constructing large, durable and fast SSD system via reprogramming 3D TLC flash memory," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2019, pp. 493–505.

[26] P. Poudel, B. Ray, and A. Milenkovic, "Microcontroller TRNGs using perturbed states of NOR flash memory cells," *IEEE Trans. Comput.*, vol. 68, no. 2, pp. 307–313, Feb. 2019.

[27] Y. Yamaga, C. Matsui, Y. Sakaki, A. Kobayashi, and K. Takeuchi, "Real usage-based precise reliability test by extracting read/write/retention-mixed real-life access of NAND flash memory from system-level SSD emulator," in *Proc. IEEE Int. Rel. Phys. Symp. (IRPS)*, Apr. 2017, pp. PM12.1–PM12.5.

[28] J.-M. Sim and Y.-H. Song, "Asymmetric read bias for alleviating cell-to-cell interference in 3D NAND flash memory," in *Proc. IEEE Region Symp. (TENSYMP)*, Aug. 2021, pp. 1–4.

[29] L. Bai, M. Wang, and J. Yi, "Design of NOR FLASH data read-write controller based on FPGA," in *Proc. 7th Int. Symp. Mechatronics Ind. Informat. (ISMII)*, Jan. 2021, pp. 104–110.

[30] T. Nakamura, Y. Deguchi, and K. Takeuchi, "9.1x error acceptable adaptive artificial neural network coupled LDPC ECC for charge-trap and floating-gate 3D-NAND flash memories," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Apr. 2018, pp. 1–4.

[31] C. Ma, Z. Zhou, L. Han, Z. Shen, Y. Wang, R. Chen, and Z. Shao, "Rebirth-FTL: Lifetime optimization via approximate storage for NAND flash memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, early access, Oct. 26, 2021, doi: 10.1109/TCAD.2021.3123177.

[32] S. T. Klein, S. Saadia, and D. Shapira, "Forward looking Huffman coding," *Theory Comput. Syst.*, vol. 65, no. 3, pp. 593–612, Apr. 2021.

[33] A. Fruchtman, Y. Gross, S. T. Klein, and D. Shapira, "Weighted adaptive Huffman coding," in *Proc. Data Compress. Conf. (DCC)*, Mar. 2020, p. 368.

[34] S. Tanakamaru, H. Yamazawa, T. Tokutomi, S. Ning, and K. Takeuchi, "19.6 hybrid storage of ReRAM/TLC NAND flash with RAID-5/6 for cloud data centers," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 336–337.

[35] J. Moon and S. Lee, "Design of H.264/AVC entropy decoder without internal ROM/RAM memories," in *Proc. 3rd Int. Symp. Commun., Control Signal Process.*, Mar. 2008, pp. 1464–1467.

[36] M. Sharafi, F. Fotouhi-Ghazvini, M. Shirali, and M. Ghassemian, "A low power cryptography solution based on chaos theory in wireless sensor nodes," *IEEE Access*, vol. 7, pp. 8737–8753, 2019.

[37] X.-J. Tong, Z. Wang, Y. Liu, M. Zhang, and L. Xu, "A novel compound chaotic block cipher for wireless sensor networks," *Commun. Nonlinear Sci. Numer. Simul.*, vol. 22, nos. 1–3, pp. 120–133, May 2015.

[38] X.-Y. Wang and Q. Yu, "A block encryption algorithm based on dynamic sequences of multiple chaotic systems," *Commun. Nonlinear Sci. Numer. Simul.*, vol. 14, no. 2, pp. 574–581, Feb. 2009.

YANAN ZHAO received the B.E. degree in the Internet of Things Engineering from Qufu Normal University, China, in 2020. She is currently pursuing the M.A.Eng. degree with the Faculty of Information Technology, Beijing University of Technology, China. Her research interests include security, privacy, and federated learning.

NONG SI (Member, IEEE) received the M.S. degree in electrical engineering from the Blekinge Institute of Technology, Sweden, and the Ph.D. degree from the Electronic Engineering Department, Beijing University of Technology, China. His research interests include security, privacy, and communication networks. He is a member of the IET and CCF.

YU SUN received the B.E. degree in telecommunication engineering from Anhui Polytechnic University, China, in 2021. She is currently pursuing the M.A.Eng degree with the Faculty of Information Technology, Beijing University of Technology, China. Her research interests include security, privacy, and federated learning.

XIN GAO received the M.E. degree in automation engineering from the Artificial Intelligence and Automation Department, Beijing University of Technology, China, in 2003. His research interests include embedded systems and wireless communications.

HAOPENG TONG is currently pursuing the B.E. degree in telecommunication engineering with the Faculty of Information Technology, Beijing University of Technology, China. His research interests include information systems and telecommunication networks.

GENG YUAN is currently pursuing the degree with the Faculty of Natural Science, Kristianstad University, Sweden. He also studied and worked with the Blekinge Institute of Technology and Lund University, Sweden. His research interests include the algorithm, applied machine learning, and statistical learning for data science. In 2007, he was awarded the Runner-Up Prize of the International Young Design Entrepreneur of 2007 by British Council.