A Chain Key Model Based on Message Hash Chain

Wenbao Jiang [*] , Jin Ding , Yongpan Wang , Huiming Du

Posted Date: 28 January 2025

doi: 10.20944/preprints202501.2021.v1

Keywords: key security; chain key; key hash chain; Salsa; message hash chain; hash function

Article

A Chain Key Model Based on Message Hash Chain

Wenbao Jiang*, Jin Ding, Yongpan Wang and Huiming Du

Department of Information Security, Beijing Infomation Science and Technology University, Beijing 102206, China

* Correspondence: jiangwenbao@bistu.edu.cn; Tel:13910027716

Abstract: To address the issue of key lifespan in key security, a chain key model based on a message hash chain is proposed. This model breaks away from the traditional scheme of generating random keys using random number generators, by using plaintext features as a source of randomness, and by generating the next key in conjunction with the current key, thus achieving key updates on a per-packet basis. The verification of the message hash chain node values ensures the consistency of the keys. The model utilizes messages and hash functions to enhance the randomness and secrecy of the keys. Security analysis shows that the complexity of cracking any key is $O(2^n)$. The experimental results indicate that, compared to the IPsec system, this model significantly reduces the lifespan of keys. Furthermore, the chain key generated by this model demonstrates good randomness in Hamming distance, autocorrelation coefficient, and entropy, thereby meeting the requirements for key randomness in secure communication. By introducing this model to generate the seed keys for the Salsa sequence cipher, the security and randomness of the Salsa sequence cipher are enhanced, increasing its complexity from the original $O(2^n)$ to $O(m2^{2^n})$, and when the transmitted message is 1400 bytes, the encryption efficiency is improved by more than 35% compared to the use of AES encryption in IPsec.

Keywords: key security; chain key; key hash chain; Salsa; message hash chain; hash function

1. Introduction

Nowadays, various encryption protocols require the two communicating parties to negotiate and update the encryption key within a specified time frame. This practice helps prevent the key from being compromised or decrypted due to prolonged use of the same key, such as in the IPsec protocol. IPsec achieves data security by encrypting every IP layer data packet, thereby realizing secure transmission at the network layer and preventing confidential data from being intercepted and eavesdropped by intermediaries. However, due to its extensive protocol configuration and complex negotiation methods [1], IPsec is typically not used in IoT embedded devices with very limited device resources. In addition, the two communicating parties usually need to negotiate the Diffie-Hellman (DH) key periodically, for example, once every hour. This short key period reduces the risk of key cracking. However, this strategy cannot indefinitely shorten the lifecycle of the key indefinitely, because frequent key agreements will increase computational and communication costs, ultimately leading to decreased communication efficiency.

Shannon demonstrated that perfect secrecy can only be achieved through the use of one-time keys, such as in the one-time pad. Even if the key is compromised, the encrypted data remains indecipherable without the specific one-time key used for encryption [2]. To approach this theory, many scholars have proposed numerous schemes, all seeking support for this theory. For example, [3] uses a commonly stored random database, where true random number factors are introduced from the outside each time and applied in the random database for retrieval and synthesis operations to generate the encryption key for each encryption; [4] proposed a method of generating new keys by shifting the initial key, grouping, and then performing cyclic confusion followed by hashing; [5] uses a pseudo-random number generator to produce a seed key for the sequence cipher, which is then encrypted with the main key and sent to the recipient along with the ciphertext. To better utilize the transmitted messages, [6] uses the transmitted plaintext and Fibonacci matrices to update the keys of both parties; synchronous sequence ciphers utilize the unpredictability of transmitted ciphertexts to increase the randomness of the sequence [7]; [8] uses the keystream generated by the RC4 algorithm as the encryption subkey for DES, solving the problem of interdependence of DES subkeys. Most of these models lack authentication functions and key synchronization mechanisms, making them vulnerable to man-in-the-middle attacks; some models have high spatial complexity and time complexity, making them less portable and scalable than IPsec. To better utilize the messages sent, [9] proposed the scheme of Automatic Variable Key (AVK). In AVK, the key needs to change automatically after each encrypted data exchange, either by generating a new key in conjunction with the plaintext or by both parties reaching a certain agreement to perform a special transformation on the initial key after sending and receiving data packets to generate the key needed for the next encryption and decryption. [10] proposed two new methods based on the AVK model: the Computation and Shift AVK (CSAVK) and the Alternating and Shift AVK (ASAVK). Subsequently, they introduced the Decimal Shift AVK (DSAVK) in [11], a scheme that enhances key variation through decimal digit shifting. [12] proposed the Divided AVK (DAVK), which achieves dynamic key adjustment through positional operations. Additionally, [13] presented a new Shift AVK model based on the concept of the Most Significant Bit, referred to as the Most Significant Bit AVK (AVKMSB). Regardless of the scheme, the objective is to establish specific rules that ensure the relationship between the generated key and the previous key is as complex as possible. In addition, [14] proposed a memory value-based chain key model. During the communication process, the session key can be adjusted in real-time based on the content of the message, significantly enhancing the security of the communication. However, the overhead primarily stems from the computation of memory values using a cryptographic accumulator, and the selection of the cryptographic accumulator directly impacts the performance of the scheme.

The main goal of this article is to enhance security by generating a unique key for each transmitted data packet, thereby increasing the unpredictability of the keys. This paper propose a chain key model based on message hash chain. The model achieves data transmission where keys change on a per-packet basis without the need for additional negotiation between the communicating parties. Through theoretical and experimental analysis, the keys generated by this model tend to be complex in terms of security, conform to random sequences in terms of randomness, and are significantly superior to the AES-encrypted IPsec protocol in terms of functionality and efficiency.

Related Work

2.1. Message Hash Chain

[15] addresses the lack of inherent security mechanisms in TCP/IP and the inefficiency of additional security technologies such as IPsec, proposing a signature authentication method based on the message hash chain. The communicating parties ensure the integrity of the message sequence through the message hash chain; when performing message signature authentication, the communicating parties only need to sign and authenticate data messages at certain intervals to ensure the integrity and non-repudiation of multiple messages, significantly improving the efficiency of secure message transmission.

Assuming the sent data is m_1, m_2, m_3 ..., and the pre-shared secret key is $prekey$, the logic is as follows:

(1) Calculate the initial node value of the message hash chain $hc_0 = hash(prekey)$.

(2) For each message m_i, it is taken out in sequence and concatenated with the last node value hc_{i-1} of the message hash chain. The hash function is then used to calculate $hc_i = hash(m_i \| hc_{i-1})$.

(3) When the last message is taken out, the complete message hash chain is obtained, with node values given by:

$$hc_i = \begin{cases} hash(m_i \,||\, hc_{i-1}) & i \geq 1 \\ hash(prekey) & i = 0 \end{cases} \tag{1}$$

2.1 *Salsa algorithm*

The Salsa algorithm is an efficient stream cipher [16], which is particularly suitable for environments with limited resources, such as embedded devices. It constructs a pseudorandom function based on fundamental operations like XOR, addition, and circular shift. A single set of round cycles can generate a 64-byte keystream. The input matrix is as follows:

$$X = \begin{pmatrix} \sigma_0 & k_0 & k_1 & k_2 \\ k_3 & \sigma_1 & n_0 & n_1 \\ \mathrm{c}_0 & c_1 & \sigma_2 & k_4 \\ k_5 & k_6 & k_7 & \sigma_3 \end{pmatrix} \tag{2}$$

Each parameter is a 32 bit binary number, where σ_i represents constants, k_i represents the key, n_i, c_i represent the initial vector and the time identifier, respectively.

The round function is composed of several basic operations and functions, including XOR, circular shift, modular addition, and other fundamental operations, as well as basic transformation functions such as row transformation $z = row(y)$, column transformation $z = col(y)$ and other functional transformations. A round of transformation can be viewed as a salsatran: $z = col(row(y))$.

Let X_n represent the matrix X after undergoing n Salsa transformations. The Salsa20 keystream block generated after one transformation is given by:

$$block = \begin{pmatrix} s_0 & s_1 & s_2 & s_3 \\ s_4 & s_5 & s_6 & s_7 \\ s_8 & s_9 & s_{10} & s_{11} \\ s_{12} & s_{13} & s_{14} & s_{15} \end{pmatrix} = X + X_{10} \tag{3}$$

3. Chain Key Model

3.1. *Symbol Description*

The main symbols used in the chain key model are explained in Table 1:

Table 1. Explanation of Key Symbols in Chain Key.

Symbol	Description
$prekey$	The initial key negotiated and allocated by the communicating parties through some secure protocol.
k_i	The key required for encrypting the i-th group of plaintext, which is also the i-th node value in the chain key.
m_i	The i-th group of plaintext.
c_i	The ciphertext corresponding to the i-th group of plaintext m_i .
$c_i = E_{k_i}(x)$	Encrypting x with key k_i to obtain the ciphertext c_i .
$m_i = D_{k_i}(x)$	Decrypting x with key k_i to obtain the plaintext m_i .
$hash_1(x)$	Hashing x for generating node values in the chain key.
$hash_2(x)$	Hashing x for generating node values in the message chain.
hc_i	The i-th node value in the message hash chain.
$mac_i = hash(m_i)$	The message authentication code (MAC) corresponding to the plaintext m_i . In this paper, the MAC is implemented as the hash value of m_i .

3.1 *Model Introduction*

Let $\{0,1\}^*$ represent the set of all finite strings, the non-empty set of ciphertexts be C, the non-empty set of plaintexts be M, and the key set be $\{0,1\}^k$, where $k\geq 1$, the non-empty set of message hash chain node values is HC. There exist two secure hash $hash_1:\{0,1\}^* \to \{0,1\}^k$, $hash_2:\{0,1\}^* \to HC$. Before data transmission, both parties determine the pre-shared key $prekey$ through DH key exchange. For any $m\in M$, $prekey \in \{0,1\}^*$, the initial key is generated from this key through the hash function, i.e. $k_1 = hash_1(prekey \| prekey)$. This model mainly consists of three algorithms with encryption and decryption capabilities and two algorithms with key synchronization capabilities.

Seed Key Generation Algorithm $Gen:(k_i, hc_i, 1^l):(k_i, hc_i) \to k_{i+1}$. It takes as input the security parameter 1^l and a pre-shared key $prekey$. Initially, the first key k_1 is derived using the formula $k_1 = hash_1(prekey \| prekey)$, while the initial hash chain tail node value hc_0 is computed as $hc_0 = hash_2(prekey)$. At each subsequent step, the hash chain tail node value hc_i is updated as $hc_i = hash_2(mac_i \| hc_{i-1})$, and the next key k_{i+1} is generated using the expression $k_{i+1} = hash_1(k_i \| hc_i)$, where $i\geq 1$. The algorithm outputs the newly generated key k_{i+1}.

The generation method is illustrated in Figure 1.

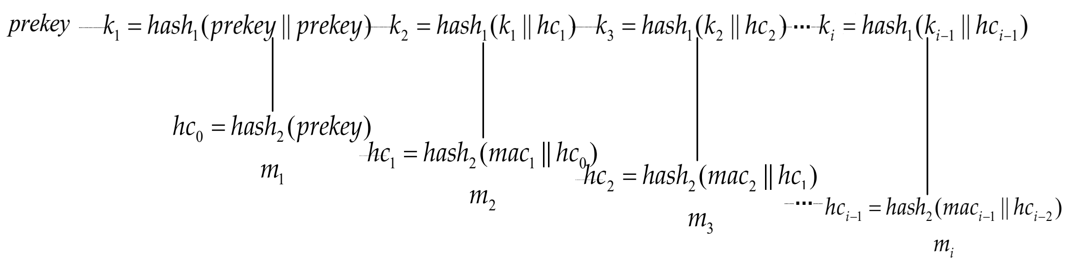

$prekey$
$k_1 = hash_1(prekey \| prekey)$
$k_2 = hash_1(k_1 \| hc_1)$
$k_3 = hash_1(k_2 \| hc_2)$
$k_i = hash_1(k_{i-1} \| hc_{i-1})$
$hc_0 = hash_2(prekey)$
m_1
$hc_1 = hash_2(mac_1 \| hc_0)$
m_2
$hc_2 = hash_2(mac_2 \| hc_1)$
m_3
$hc_{i-1} = hash_2(mac_{i-1} \| hc_{i-2})$
m_i

Figure 1. Logic Diagram of Chain Key Construction.

Data Encryption Algorithm $enc: k_i \times \{0,1\}^* \times HC \to C \cup \{*\}$. Given a plaintext message $m_i \in M$, the algorithm first computes the message hash chain tail node value $hc_i = hash_2(mac_i \| hc_{i-1})$, where mac_i is the message authentication code associated with m_i. Using the current key k_{i-1} and the computed hc_i, the new encryption key k_i is derived as $k_i = hash_1(k_{i-1} \| hc_{i-1})$. Subsequently, the plaintext m_i and the authentication code mac_i are encrypted together with the key k_i, producing the ciphertext $c_i = E_{k_i}(m_i, mac_i)$. Finally, the algorithm outputs the encrypted ciphertext c_i.

Data Decryption Algorithm $dec: k_i \times \{0,1\}^* \to M \cup \{*\}$. Given a ciphertext $c_i \in C$, the algorithm first retrieves the encryption key k_i by computing $k_i = hash_1(k_{i-1} \| hc_{i-1})$, where hc_{i-1} is the message hash chain tail node value associated with the previous message. Using the derived key k_i, the ciphertext c_i is decrypted as $D_{k_i}(c_i) = (m_i, mac_i)$, yielding the plaintext m_i and its corresponding authentication code mac_i. The algorithm outputs the plaintext m_i and the associated authentication code mac_i.

For the correctness of the aforementioned three algorithms, the following explanations are provided:

a. One-wayness of hash functions: For all $m\in M$, the hash value hc is computed as $hc = hash_2(m)$, where $hc \in HC$ and $m \in \{0,1\}^*$. It is computationally infeasible to find the original input m from the hash value hc. Namely, for any input m, the hash function $hash_2(m)$ produces an output hc, but given hc, it is practically impossible to compute m in a reasonable amount of time.

b. Consistency of key generation: For all $k_i \in \{0,1\}^k$, the next key is $k_{i+1} = hash_1(k_i \| hc_i)$, where $hc_i \in \{0,1\}^*$.

c. Effectiveness of the encryption process: For all $k_i \in \{0,1\}^k$ and $m \in \{0,1\}^*$:

- If $m \notin M$, then $enc_{k_i}(m, hc_m) = *$.
- If $m \in M$, then $enc_{k_i}(m, mac_m) \in C$, and the decryption satisfies $D_{k_i}(E_{k_i}(m, mac_m)) = (m, mac_m)$.

d. Security of key updates: For all $k_i \in \{0,1\}^k$, the updated key satisfies $k_{i+1} = Gen(k_i, hc, 1^l) \in \{0,1\}^k$.

To ensure that the keys are not compromised during transmission due to accidents or malicious attacks by adversaries, this model incorporates a key synchronization module, which includes a verification algorithm and a retransmission algorithm.

Verification Algorithm: $Ver:(c_i, k_i) \rightarrow mac_i'$. After the data decryption algorithm processes the ciphertext c_i using the decryption key k_i, it retrieves the plaintext m_i and the transmitted authentication code mac_i: $D_{k_i}(c_i) = (m_i, mac_i)$. Subsequently, the verification algorithm computes a new authentication code mac_i' over the plaintext m_i using a secure hash function: $mac_i' = hash(m_i)$. Finally, the algorithm compares the computed mac_i' with the transmitted mac_i to verify the integrity and authenticity of the message:

- If $mac_i' = mac_i$, the verification is successful, and the plaintext m_i is consideres valid.
- If $mac_i' \neq mac_i$, the verification fails, and an error message is returned (e.g., "MAC verification failed").

Retransmission Module: $Rsend:(i, ASK[max]) \rightarrow (ASK[i], ASK[i-1] = true)$. Upon receiving a verification failure ($mac_i' \neq mac_i$), the module identifies the current retransmission request array $ASK[max]$ for the associated transmission window. To ensure proper retransmission, the module updates the request array ASK by setting the retransmission flags for the current packet $ASK[i]$ and the previous packet $ASK[i-1]$ to true. This ensures that both the current and preceding packets are included in the retransmission request.The updated retransmission request array ASK is then transmitted to the sender to initiate the retransmission process for the specified packets.

The relationships among the various modules in this model are depicted in Figure 2.

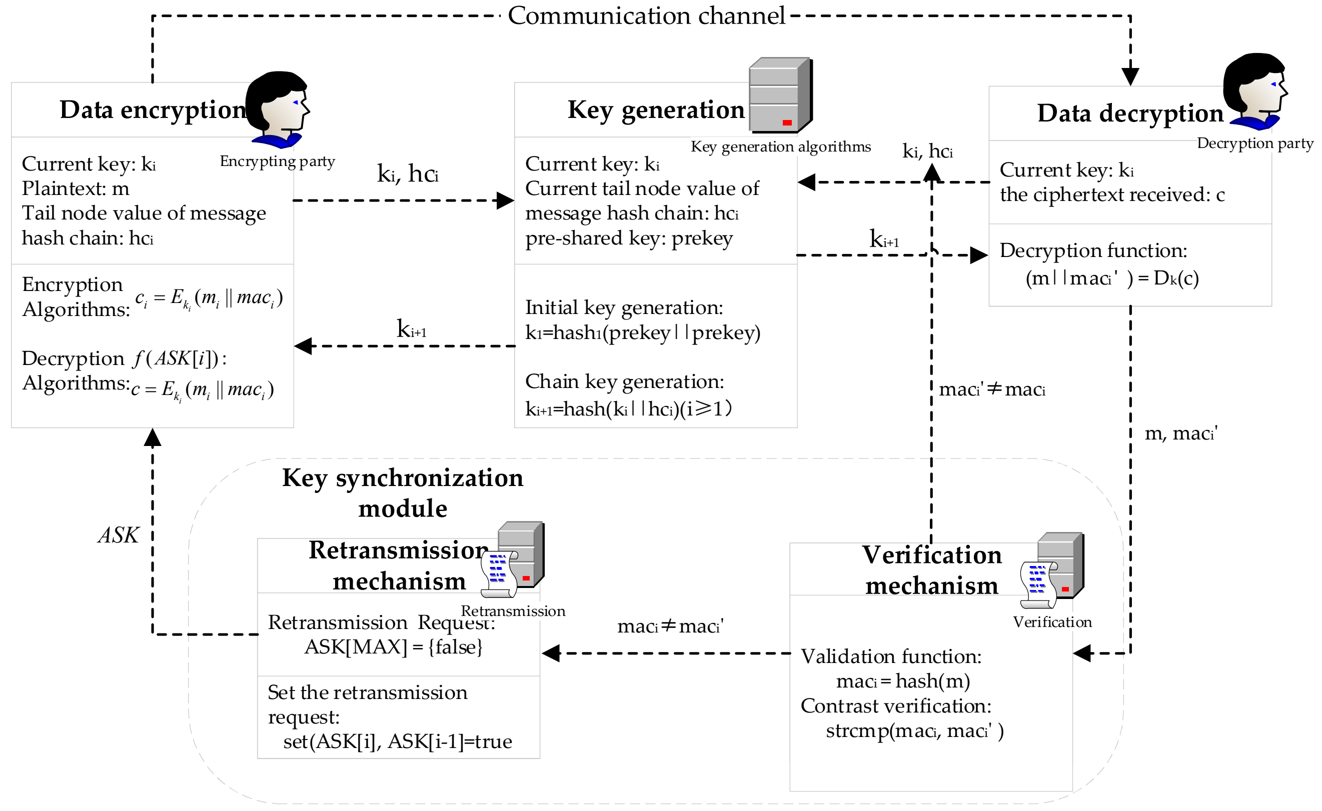

Communication channel
Data encryption
Encrypting party
Current key: k_i
Plaintext: m
Tail node value of message hash chain: hc_i
Encryption Algorithms: $c_i = E_{k_i}(m_i \| mac_i)$
Decryption Algorithms: $f(ASK[i])$: $c = E_{k_i}(m_i \| mac_i)$
k_i, hc_i
k_{i+1}
Key generation
Key generation algorithms
Current key: k_i
Current tail node value of message hash chain: hc_i
pre-shared key: prekey
Initial key generation: k_1=hash$_1$(prekey||prekey)
Chain key generation: k_{i+1}=hash(k_i||hc_i)(i⩾1)
k_i, hc_i
k_{i+1}
Data decryption
Decryption party
Current key: k_i
the ciphertext received: c
Decryption function: (m||mac_i') = D_k(c)
mac_i'≠mac_i
m, mac_i'
ASK
Key synchronization module
Retransmission mechanism
Retransmission
Retransmission Request: ASK[MAX] = {false}
Set the retransmission request: set(ASK[i], ASK[i-1]=true
mac_i≠mac_i'
Verification mechanism
Verification
Validation function: mac_i = hash(m)
Contrast verification: strcmp(mac_i, mac_i')

Figure 2. The Relationships Among Modules in the Chain Key Model.

3.2. Specific Process

3.3.1. Sender Procedure

This is example 1 of an equation:

Step 1: The sender and receiver use the same pre-shared secret key $prekey$ and algorithm to compute the first key $k_1 = hash_1(prekey \| prekey)$, which serves as the initial node value of the chain key and, at the same time, as the current tail node value of the chain key.

Step 2: Use the hash function to compute the hash chain node value $hc_i = hash_2(mac_i \| hc_{i-1})$. Append this value to the end of the message chain as the new tail node value. Then encrypt both the message m_i and authentication code mac_i with the tail key $k_i(i>0)$ to generate the encrypted message $c_i = E_{k_i}(m_i \| mac_i)$, where mac_i serves as the authentication condition.

Step 3: Send the corresponding encrypted message to the receiver, who will retrieve the tail node hc_i of the report chain and the tail node k_i of the chain key. Using the hash function $hash_1$, the next key $k_{i+1} = hash_1(k_i \| hc_i)$ is calculated, adding this key to the tail node of the chain key as the new tail node value.

Step 4: Check the current communication status. If it needs to be terminated, end the process; otherwise, return to Step 2.

For the sender, the pseudocode is as follows:

Algorithm 1. Sender Process

```
Input: pre-shared key $prekey$, message $m_i$,  plaintext hash value $mac_1$
Output: ciphertext $c_i$
    1.  $set(prekey)$
    2.  $k_1 = hash_1(prekey \| prekey), hc_0 = hash_2(prekey)$
    3.  $hc_1 = hash_2(mac_1 \| hc_0)$
    4.  $c_1 = E_{k_1}(m_1 \| mac_1), send(c_1)$
    5.  $i = 2, k_2 = hash_1(k_1 \| hc_1)$
    6.  $while(m_i != null)$
    7.      $hc_i = hash_2(mac_i \| hc_{i-1})$
    8.      $c_i = E_{k_i}(m_i \| mac_i)$
    9.      $send(c_1)$
    10.     $i = i + 1$
    11.     $k_i = hash_1(k_{i-1} \| hc_{i-1})$
    12. $return$
```

3.3.2. Receiver Process

For the sender, the pseudocode is as follows:

Step 1: The receiver uses the same pre-shared key and algorithm as the sender. Using the pre-shared key $prekey$, the first key $k_1 = hash_1(prekey \| prekey)$ is calculated, which serves as the initial node of the chain key and also as the current tail node of the chain key.

Step 2: The tail key $k_i(i>0)$ of the chain key is used to decrypt the received ciphertext c_i, extracting m_i, mac_i from it. Calculate $mac_i' = hash(m_i)$, if $mac_i' = mac_i$, verification is successful. Then add $hc_i = hash_2(mac_i \| hc_{i-1})$ to the report chain, setting it as the new tail node of the report chain.

Step 3: Retrieve the tail nodes of both the report chain and the chain key. Use the hash function $hash_1$ to calculate the next key $k_{i+1} = hash_1(k_i \| hc_i)$, and add it to the tail of the chain key as the new tail node value k_{i+1}.

Step 4: If a termination request is received, end the process; otherwise, wait to receive the next report and proceed to Step 2.

For the receiver, the pseudocode is as follows:

Algorithm2. Receiver Process

```
Input: pre-shared key $prekey$, ciphertext $c_i$, key $k_i$
Output: message $m_i$
1.  $set(prekey)$
2.  $k_1 = hash_1(prekey \| prekey), hc_0 = hash_2(prekey)$
3.  $receive(c_1)$
4.  $m_1 \| mac_1 = D_{k_1}(c_1)$
5.  $if(mac_i != hash(m_i))$
6.      $return\ error$
7.  $i = 2, k_2 = hash_1(k_1 \| hc_1)$
8.  $while(c_i != null)$
9.      $receive(c_i), (m_i \| mac_i) \leftarrow D_{k_i}(c_i)$
10.     $if(mac_i != hash(m_i))$
11.         $send(auth\ error)$
12.         $continue$
13.     $i = i + 1$
14.     $k_i = hash_1(k_{i-1} \| hc_{i-1})$
15. $return$
```

Taking an example from one of the transmissions during the data transfer process, the message reception process and the updating process of the two hash chains are illustrated in Figure3.

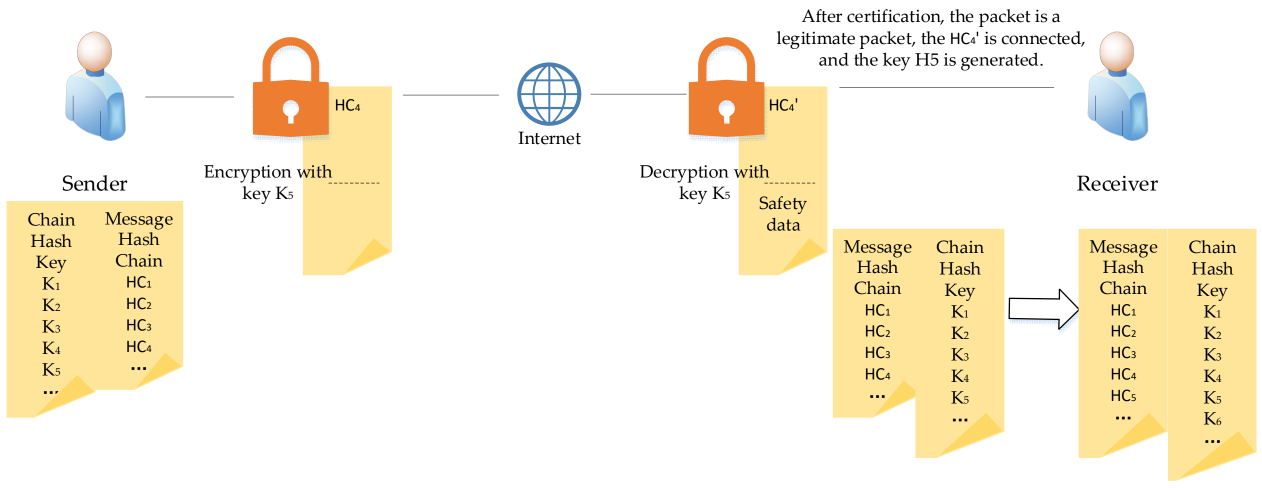

After certification, the packet is a legitimate packet, the HC₄' is connected, and the key H5 is generated.
Sender
Encryption with key K₅
HC₄
Internet
Decryption with key K₅
HC₄'
Safety data
Receiver
Chain Hash Key K₁ K₂ K₃ K₄ K₅ …
Message Hash Chain HC₁ HC₂ HC₃ HC₄ …
Message Hash Chain HC₁ HC₂ HC₃ HC₄ …
Chain Hash Key K₁ K₂ K₃ K₄ K₅ …
Message Hash Chain HC₁ HC₂ HC₃ HC₄ HC₅ …
Chain Hash Key K₁ K₂ K₃ K₄ K₅ K₆ …

Figure 3. The process of updating the key hash chain.

It is worth mentioning that after receiving the message, the receiver determines whether the message has been tampered with by verifying if $mac_i \overset{?}{=} mac_i'$, additionally, the dynamic update of the chain key's tail node ensures that each transmission uses an independent key, thereby reducing the risk of replay attacks and tampering. The tail node of the chain key, as one of the authentication conditions, helps the receiver detect anomalies during verification. Therefore, it can be concluded that the message received by the receiver is error-free.

3.2. Chain key and Self-Synchronizing Stream Cipher

Both self-synchronizing stream ciphers and chain keys incorporate the randomness of the plaintext into the key generation process, achieving key unpredictability to disrupt the key cycle.

For chain keys, the generation scheme for key k_i and ciphertext c_i is as follows:

- $\sigma_i = (m_1, m_2, ..., m_{i-1})$
- $z_i = g(hc(\sigma_i), k)$
- $c_i = h(z_i, m_i)$

For self-synchronizing stream ciphers, the generation scheme for the keystream z_i and ciphertext c_i is as follows:

- $\sigma_i = (c_{i-t}, c_{i-t+1}, ..., c_{i-1})$
- $z_i = g(\sigma_i, k)$
- $c_i = h(z_i, m_i)$

The difference lies in that a self-synchronizing stream cipher takes a set of t ciphertext characters $\{c_{i-t}, c_{i-t+1}, ..., c_{i-1}\}$ as parameters for the key generation function. This enables the self-synchronizing stream cipher to have a limited error propagation mechanism, allowing it to resynchronize automatically after certain transmission errors. On the other hand, although the classic chain key does not possess this type of self-synchronizing mechanism, it utilizes the report hash chain as a condition for key synchronization. This allows it to detect transmission errors and trigger key synchronization on both sides of the communication.

Considering only the relationship between the key and the plaintext, the self-synchronizing stream cipher is represented as follows:

$$k_i = F\left(\cup_{j=i}^{t} c_{j-i}, k_{i-1}\right) = F'\left(\cup_{j=0}^{i-1} c_j, k\right) \tag{4}$$

The chain key is represented as follows:

$$k_i = G(m_{i-1}, k_{i-1}) = G'\left(\cup_{j=0}^{i-1} m_j, k\right) = G'\left(\cup_{j=0}^{i-1} D_{k_j}(c_j), k\right) \tag{5}$$

By analyzing the above expression, it can be observed that in the process of generating a new key, the self-synchronizing stream cipher and the chain key differ not only in the key generation function but also in the internal parameters used. The parameters of the self-synchronizing stream cipher mainly depend on the previous key (or internal state) and the preceding t ciphertext characters. From an overall perspective, it can be seen as a new key generated by the combined effect of the initial state and all prior ciphertexts. The issue is that if an attacker obtains the internal state of the automaton and a number of ciphertext characters, they can also decrypt the message, meaning that the self-synchronizing stream cipher lacks forward secrecy.

In comparison, a chain key not only attempts to recover the previous key but also requires the previous key to decrypt ciphertext c_j in the decryption process. Assuming the decryption complexity is $O(2^n)$, it adds an additional targeted decryption of k compared to the self-synchronizing stream cipher with a complexity of $O(2^n)$, making the chain key more challenging with a complexity of $O(2^{2n})$.

A comparison between chain key and self-synchronizing stream cipher is shown in Table 2.

Table 2. Comparison between Chain Key and Self-Synchronizing Stream Cipher.

Comparison Metric	Self-Synchronizing Stream Cipher	Chain Key Model
Key Synchronization	Self-synchronizing	Computation Synchronization
Plaintext Participation Form	Ciphertext	Plaintext

Forward Secrecy	Yes	Yes
Backward Secrecy	No	Yes
Complexity	$O(2^n)$	$O(2^{2n})$

4. Salsa Stream Cipher Optimization Based on Chain Key

Currently, there are no effective attacks against the Salsa stream cipher, and analysis of the Salsa round function has reached up to 5 rounds. At this stage, the Salsa stream cipher can be considered secure. Due to the simplicity of the stream cipher's initialization process, which only requires assigning values to 16 32-bit words, the Salsa stream cipher can easily update the key at any time during encryption. This prevents the same key from being exploited by attackers over a long period. Regularly updating the key reduces the probability of successful brute-force attacks or other cryptographic analysis attacks, as attackers need to continuously restart attempts to crack the new keys.

As shown in (2), the initialization state of the Salsa stream cipher generates a 64-byte keystream in each round. Let its round function be F, with each round's input being the output of the previous round. The 64-byte block generated in the *i*-th set is a key block $block_i$, therefore:

$$block_i = \begin{cases} F(X) & i = 1 \\ F(block_{i-1}) & i > 1 \end{cases} \tag{6}$$

The key hash chain and report hash chain generated in this paper are used to reconstruct the Salsa stream cipher. When re-authentication is triggered, the Salsa stream cipher will no longer generate the next key block $block_i$ according to (6), but will instead initialize it based on the initial state as:

$$X(k_i, hc_{i-1}) = \begin{pmatrix} \sigma_0 & k_{i0} & k_{i1} & k_{i2} \\ k_{i3} & \sigma_1 & hc_{(i-1)0} & hc_{(i-1)1} \\ hc_{(i-1)2} & hc_{(i-1)3} & \sigma_2 & k_{i4} \\ k_{i5} & k_{i6} & k_{i7} & \sigma_3 \end{pmatrix} \tag{7}$$

In Equation (7), k_{it} and $hc_{(i-1)t}$ are determined by the tail node values k_i and hc_{i-1} of the key hash chain and message hash chain, respectively. This ensures that after a certain execution period, the calculation period of Salsa is reset, and the calculation rules change. The workflow of the modified Salsa stream cipher is as follows:

Step 1: Input the initial key $prekey$, calculate the first key $k_1 = hash_1(prekey \| prekey)$, and set the initial node of the report hash chain $hc_0 = hash_2(prekey)$, with $i = 1$.

Step 2: Generate the keystream S_i of length $l = strlen(m_i)$ according to the rule in Equation (5), $S_i = Salsa\left(k_i, hc_{i-1}\right)^l$.

Step 3: Calculate $hc_i = hash_2(hc_{i-1} \| m_i)$, $k_{i+1} = hash_1(k_i \| hc_i)$.

Step 4: Based on k_{i+1} and hc_i, set $i = i + 1$, and return to execute Step 2.

The modified Salsa stream cipher now has key synchronization functionality. When the hc values of both parties are not synchronized, both sides can detect and correct it, thereby preventing a man-in-the-middle attack from maliciously altering keys to cause synchronization issues.

5. Security Analysis

5.1. Symbol Description

Related-key attacks are a type of key attack that exploits specific properties in key expansion schemes, studying relationships between different keys to obtain information about one or more keys. Attackers typically use various techniques to reduce the complexity of specific attacks by identifying correlations between keys. This paper provides a proof of the security of the chain key under the Random Oracle Model (ROM).

References [17, 18] highlight the special properties of hash functions and conclude that strongly obfuscated hash operations are statistically unbiased. Data processed by a strongly obfuscated hash function is completely scrambled, and under the Random Oracle Model (ROM), the data before and after processing can be considered, with high probability, as outputs of a random function, meaning that non-randomness is negligible.

In key updates, there is usually some computational relationship between adjacent keys. To consider the impact of this relationship on the difficulty of deciphering adjacent keys, the following definitions and corollaries are provided.

Definition 1. There exist two sets of keys, where key set B is obtained from key set A through a certain complex transformation, and key set B cannot be used to derive key set A. In this case, key sets A and B are said to have a derivative relationship. The process of derivation is called the derivation algorithm. The keys in key set A are referred to as master keys, and the keys in key set B are referred to as derived sub-keys.

Lemma 1. For master key set A and derived sub-key set B that have a derivative relationship, if key set B is derived from key set A using a derivation algorithm modeled within the Random Oracle Model (ROM), then the difficulty of breaking key set B is equivalent to that of breaking key set A, assuming the derivation algorithm is secure in the ROM.

Proof. Given that key set B is derived from key set A through a derivation algorithm.

According to Definition 1, if key set A can be broken, then key set B can be obtained through the derivation algorithm. Conversely, without breaking key set A, it is not possible to obtain any additional assistance in breaking key set B. Therefore, if an attacker does not have key set A, they cannot directly obtain key set B.

For the same algorithm, key set B is derived from key set A through a complex transformation. Under the Random Oracle Model (ROM), the distribution of key set B within the key space is identical to that of key set A. Therefore, the complexity of decrypting messages encrypted with key set B is equivalent to that of decrypting messages encrypted with key set A.

From the above, when key set A is compromised, key set B is simultaneously compromised. When key set A remains secure, the difficulty of breaking key set B is equivalent to that of breaking key set A.

Q.E.D.

To consider the correlation between two sequences, we typically examine their computational indistinguishability. The following definitions are provided.

Definition 2. Computational Indistinguishability: Given two sequences, $\{X_n\}_{n\in N}$ and $\{Y_n\}_{n\in N}$, if there is no effective algorithm to distinguish between them, then the two sequences are said to be computationally indistinguishable.

Definition 3. Statistical Distance: Let $X=\{X_n\}_{n\in N}$ and $Y=\{Y_n\}_{n\in N}$ be two populations. Let $\varDelta(n)$ represent the statistical distance between the two populations X and Y; then $\varDelta(n)$ is defined as follows.

$$\varDelta(n)=\frac{1}{2}\sum_{a}|\,pr[X_n=a]-pr[Y_n=a]\,| \tag{8}$$

Property of Statistical Indistinguishability: If $\varDelta(n)$ is negligible, then $\{X_n\}_{n\in N}$ and $\{Y_n\}_{n\in N}$ are said to be statistically close. Furthermore, if they are statistically close, then they are indistinguishable.

Based on the definitions , the following two lemmas are presented:

Lemma 2. The intermediate key k_i in the chain key and the initial key k_1 have a derivative relationship, and the difficulty for an attacker to obtain the intermediate key k_i will not be lower than $O(2^n)$, under the assumption that the derivation algorithm is secure in the Random Oracle

Model (ROM). In other words, an attacker cannot obtain the intermediate key k_i with a higher probability without knowing k_1.

The proof by contradiction is as follows:

Proof. Suppose an attacker could break the intermediate key k_i without knowing k_1. Then the attacker would need to obtain correlations between at least two intermediate keys and use these correlations to perform cryptanalysis methods, such as differential analysis. If the attacker is able to successfully break k_i, this would contradict Shannon's proof of perfect security, particularly the absolute security of the One-Time Pad (OTP). Shannon proved that if a key is completely random and used only once, the ciphertext reveals no information about the plaintext. Therefore, if an attacker can break k_i, it indicates that these keys are not computationally indistinguishable, meaning the statistical difference $\Delta(n)$ in Equation (8) is not negligible.

However, the keys used in this model are generated under the ROM (Random Oracle Model) using strongly obfuscated hash functions. Based on the properties of strongly obfuscated hashes, their output can be statistically regarded as a random permutation, meaning the correlation between keys is statistically negligible (i.e., $\Delta(n)$ is negligible). Therefore, an attacker cannot break k_i by analyzing the correlations between multiple keys, as the statistical difference between the keys is negligible. In other words, an attacker cannot break the intermediate key k_i without knowing k_1.

Q.E.D.

Lemma 3. The security of the intermediate keys in the chain key depends on the one-wayness of hash function $hash$, the randomness of the plaintext sequence $M = \{M_n\}_{n\in N}$, and the security of the pre-shared key $prekey$, and is unrelated to the state of the intermediate keys.

The proof is carried out using mathematical induction as follows:

Proof. (1) The initial key k_1 is derived from the pre-shared key $prekey$ through the hash function, i.e., $k_1 = hash_1(prekey \| prekey)$. Since no plaintext is involved, the security of the initial key depends on:

- The one-wayness of the hash function $hash$: An attacker cannot infer prekey $prekey$ from k_1.
- The confidentiality of the pre-shared key prekey $prekey$: An attacker cannot directly access prekey $prekey$.

Thus, the security of k_1 is determined by the one-wayness of the hash function, as well as the confidentiality of prekey $prekey$.

(2) Assume that the security of the key $k_i (i > 1)$ is determined by:

- The one-wayness of the hash function $hash$.
- The unpredictability of the plaintext sequence $M = \{M_n\}_{n\in N}$.
- The confidentiality of the pre-shared key prekey $prekey$.

(3) The key k_{i+1} is generated as $k_{i+1} = hash_1(k_i \| hc_i)$, where $hc_i = hash_2(m_i \| hc_{i-1})$, Since the security of $k_i (i > 1)$ is guaranteed by the inductive hypothesis, and hc_i depends on the plaintext m_i and the hash function $hash_2$, the security of k_{i+1} is determined by:

- The one-wayness of the hash functions $hash_1$ and $hash_2$.
- The unpredictability of the plaintext sequence $M = \{M_n\}_{n\in N}$.
- The confidentiality of the pre-shared key prekey $prekey$.

Therefore, k_{i+1} also satisfies Lemma 3.

(4) From (1), (2), and (3), it follows that the security of all keys in the chain key depends on the one-wayness of the hash function $hash$, the unpredictability of the plaintext sequence $M = \{M_n\}_{n\in N}$, and the confidentiality of the pre-shared key $prekey$. Thus, Lemma 3 holds.

Q.E.D.

5.1. Key Extensibility Security

In a chain key model, whether an attacker can derive all subsequent keys if they obtain a key at a certain point depends on the design of the chain key generation mechanism. An ideal chain key model should possess both forward secrecy and backward secrecy. The following analysis addresses this.

Definition 4. Forward Secrecy: If the designed key generation model ensures that the discovery of a key does not compromise previous session keys, even if an attacker obtains a key at a specific point in time, they cannot decrypt previous communication records.

Lemma 4. The keys generated by a chain key model based on a hash algorithm possess forward secrecy.

Proof by contradiction:

Proof. (1) In a certain communication process, the key update sequence between the communicating parties is $K = \{k_1, k_2, k_3 ... k_{i-1}, k_i\}$. Suppose an attacker obtains the key k_i of the *i*-th communication. The key k_i is generated as $k_i = hash_1(k_{i-1} \| hc_{i-1})$.

(2) To break forward secrecy, the attacker would need to deduce k_{i-1} from k_i. However, since $hash_1$ is a one-way function, it is computationally infeasible to deduce $k_{i-1} \| hc_{i-1}$ from k_i.

(3) Therefore, the attacker cannot deduce k_{i-1} from k_i and the exposure of k_i does not compromise the security of k_{i-1}.

(4) By induction, this property holds for all previous keys in the chain. Thus, the keys generated by the chain key model based on hash algorithms possess forward secrecy.

Q.E.D.

Definition 5. Backward Secrecy: Similar to forward secrecy, backward secrecy means that even if the current key is compromised, it does not affect the security of future sessions. This is typically achieved by introducing new randomness during the key update process, making it impossible to derive future keys from the current key.

Definition 6. Key Dependency: The key update mechanism should be designed to depend on random values transmitted during the current session or session-specific information, so that even if an attacker obtains the current key, they cannot use it alone to derive the next key.

Definition 7. State Complexity: If the update mechanism includes complex state information, an attacker may need more information to derive the next key, which may not be obtainable in a single communication.

Lemma 5. The keys generated by a chain key model based on hash algorithms possess backward secrecy.

Proof by contradiction:

Proof. (1) In a certain communication process, the key update sequence between the communicating parties is $K = \{k_1, k_2, k_3 ... k_i, k_{i+1}\}$, an attacker, using some method, obtains the key k_i of the *i*-th communication.

(2) The key k_{i+1} is generated from the key k_i and hc through the hash function $hash_1$.That is, $k_{i+1} = hash_1(k_i \| hc_i)$. Suppose that breaking k_i would reveal k_{i+1}, then the attacker would need to obtain the report hash chain node hc_i through k_i.

(3) Breaking k_i would lead to the exposure of the information from the *i*-th communication, allowing the attacker to obtain the plaintext message m_i and the message verification code mac_i, Furthermore, by Corollary 4, breaking k_i does not expose k_{i-1} and hc_{i-1}, therefore, breaking k_i does not provide more information about hc_{i-1}, and the attacker cannot obtain $hc_i = hash_2(mac_i \| hc_{i-1})$. This contradicts the assumption, so the assumption does not hold. Therefore, breaking k_i does not lead to the exposure of k_{i+1}. Here, mac_i and hc_i serve as the complex state information and random values transmitted as part of the key update mechanism in the current session.

(4) Furthermore, since the generation of k_{i+2} directly involves k_{i+1}, the exposure of k_i does not affect the security of the subsequent key k_{i+1}.

(5) Therefore, the keys generated by the chain key model based on hash algorithms possess backward secrecy.

Q.E.D.

The chain key model's design makes it a standard of randomness, utilizing the hash node $hash_2(mac_i \| hc_{i-1})$, which is derived from the set of specific session information and extrinsic complex information of each report message in communication. According to Corollary 5, the attacker cannot deduce a single future key from the current session.

5.1. Security Analysis of Salsa Stream Cipher Based on Chain Key

To demonstrate the security of the Salsa stream cipher and the chain key model, we present the following security assumptions:

Definition 8. If a key-based deterministic algorithm produces an output that, given a specific key, appears random and is indistinguishable from a truly random function, it is called a Pseudorandom Function (PRF).

The security of the Salsa stream cipher relies on the following assumptions:

Assumption 1. The sequence generated by Salsa satisfies Definition 8, meaning the sequence generation algorithm of Salsa is pseudorandom.

Assumption 2. A 256-bit key is secure after sufficient rounds of iteration.

The chain key model relies on the following security assumptions:

Assumption 3. The hash function H is modeled as a random function in the Random Oracle Model (ROM), with the properties of a highly obfuscated hash function.

Assumption 4. The initial key K_0 is secure.

An attacker A attempting to break the Salsa algorithm within polynomial time would need the ability to break Assumptions 1 and 2. Under the same assumptions, if attacker A aims to break the Salsa stream cipher based on the chain key model within polynomial time, they would need to simultaneously break the security assumptions of Salsa (Assumptions 1 and 2) and the security assumptions of the chain key model (Assumptions 3 and 4). Therefore, under the same assumptions, Salsa based on the chain key model offers higher security than a standalone Salsa. Furthermore, the key update mechanism limits the impact of breaking any single seed key in the Salsa stream cipher.

For an attacker A, upon learning that Salsa uses a seed key of length n, they obtain a sequence of length $s = m * l$, where m represents the number of seed key updates in the chain model, and l represents the length of the sequence generated by each seed key. If attacking a true random sequence, the complexity of attacking the ciphertext should be $O(2^{ml})$. The complexity of attacking the Salsa sequence is $O(2^n)$, which is the difficulty of decrypting a single Salsa. The complexity based on the chain key is $O(m2^{2n})$, which is the difficulty of decrypting m Salsa sequence ciphers and m hash functions. The attack complexities for the three scenarios are shown in Table 3.

Table 3. Attack Complexity.

Keystream Source	Attack Complexity
Truly Random Sequence	$O(2^{ml})$
$Salsa$	$O(2^n)$
$hash - Salsa$	$O(m2^{2n})$

6. Experimental Analysis

6.1. Chain key Experiment Analysis

The overall experimental platform for this test is configured as follows: Windows 11 operating system, Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, and 32GB RAM. The specific experiments were conducted within a virtual machine environment.

6.1.1. Chain key Generation Efficiency

To assess the generation efficiency of the chain key model, this section investigates the number of messages exchanged between the communicating parties and the total number of message bytes transmitted within a fixed time interval during actual communication. To illustrate the communication efficiency of the chain key model, a comparison is made with the mainstream IPsec protocol to highlight the advantages of a hash-based chain key approach. These advantages are primarily in two aspects:

Firstly, the hash-based chain key model updates keys on a per-packet basis, eliminating the need for additional key negotiation during transmission, thereby significantly reducing communication overhead and improving transmission efficiency. In contrast, the IPsec protocol requires Diffie-Hellman (DH) key negotiation when necessary, which increases communication complexity and time costs.

Secondly, the chain key model utilizes hash algorithms to generate keys, and the computation speed of hash algorithms is typically fast, enabling efficient support for high-frequency data transmission and meeting real-time communication requirements. DH key negotiation, on the other hand, is limited by algorithm complexity and computational resources, and shortening the update cycle may affect efficiency.

Due to these two factors, the chain key model outperforms the DH-negotiated IPsec communication protocol in terms of the number of messages communicated within a unit of time and bitrate. In this experiment, UDP data transmission was used, with specific message lengths randomly within 1400 bytes, including a 32-byte MAC value and a 4-byte sequence value. The experimental testing process involved the chain key model updating keys on a per-packet basis, while the DH key negotiation cycle in IPsec was set to 10 seconds. The specific test data is shown in the following Table 4.

Table 4. Measurement Results of Message Volume for Chain key and DH Key Negotiation Cycles.

Communication Mode		1s	10s	100s
Chain key	Message Transmission Volume/Packets	1675	16522	151062
	Total Message Length/Bytes	1112816	10744416	119830932
	Bit Rate/bps	8902528	8595532.8	9586478.56
DH Key Negotiation	Message Transmission Volume/Packets	391	12365	114020
	Total Message Length/Bytes	310231	9014271	92014231
	Bit Rate/bps	2481848	7211416.8	7361138.48

From the table above, it is evident that the communication scheme employing the chain key model achieves a higher bit rate compared to the standard IPsec communication scheme. Moreover, frequent key negotiations increase the complexity of key management, requiring each new key to be securely stored to ensure its availability and security. In contrast, the chain key model locally stores the message hash chain used for signing, fulfilling dual purposes with a single chain, significantly reducing the cost and risk of key management.

6.1.3. Chain key Randomness Test

This section of the experiment demonstrates the randomness of the keys generated by the chain key model from three aspects: Hamming distance, autocorrelation coefficient, and entropy. Among them, the experiments on autocorrelation coefficients and entropy refer to reference [14].

(1) Hamming Distance

In analyzing the randomness between two adjacent keys, the Hamming distance is commonly used to estimate the randomness between two adjacent data points [19,20]. This paper compares the randomness of keys generated by the chain key model with several one-time pad algorithms, presenting the Hamming distances of the random keys produced by each algorithm.

During message transmission, special messages (such as TCP three-way handshake protocol messages) have a significant impact on message identification fields, such as cookie values. To amplify the demonstration of the impact of special content on the key generation of each model, this experiment sets the initial key to 0x00000000 and the plaintext array to {0x00010203, 0x04050607, 0x08090A0B, 0x0C0D0E0F, 0x10111213…}, totaling 1000 sets of plaintext. Under such conditions where the plaintext has a certain regular distribution, the randomness deficiency due to the algorithm's own logic issues can be magnified, making it easier to illustrate the problem.

The randomness of each algorithm, after computation, is shown in Tables 5 and 6. The horizontal numbers represent different Hamming distances, while the vertical ones represent different algorithms. The tables count the number of times the Hamming distance between adjacent keys is a certain value during the key generation process.

Table 5. Hamming Distance Distribution 1.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
AVK	0	0	0	0	16	0	0	0	96	0	0	0	238	0	0	0	313
ASAVK	0	0	0	0	266	0	0	0	330	0	0	0	158	0	0	0	155
DSAVK	0	0	0	0	1	0	0	0	5	7	0	22	58	68	159	85	185
DAVK	0	0	0	0	3	0	2	0	7	0	1	0	98	0	232	0	285
AVKMSB	0	0	0	0	1	1	0	0	6	4	7	17	69	72	141	78	194
Chain key	0	0	0	0	0	0	0	0	0	6	4	14	48	66	106	126	153

Table 6. Hamming Distance Distribution 2.

	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
AVK	0	0	0	231	0	0	0	91	0	0	0	15	0	0	0	0
ASAVK	0	0	0	76	0	0	0	15	0	0	0	0	0	0	0	0
DSAVK	89	137	33	75	20	28	4	1	1	1	1	0	0	0	0	0
DAVK	0	227	0	93	0	21	0	1	0	0	0	0	0	0	0	0
AVKMSB	73	139	58	78	19	20	7	5	1	0	0	0	0	0	0	0
Chain key	179	117	76	56	36	9	0	4	0	0	0	0	0	0	0	0

From Tables 5 and 6, it can be observed that AVK, ASAVK, and DAVK exhibit peculiarities in key transformation under specific plaintexts, with Hamming distances only present in particular values. The fundamental reason is that their computation process involves continuous XOR operations on the plaintext. When the plaintext has special characteristics, it can lead to two or more XOR results canceling each other out, resulting in key leakage and other issues. Moreover, the Hamming distances of these algorithms are mostly distributed in the lower ranges. This indicates that the randomness in key generation decreases as the regularity of the plaintext increases.

If n represents the number of identical bits between two consecutive keys, then the number of different bits is 32-n. Given the previous key and n,the number of possible values for the subsequent key is C_{32}^{n}, which means selecting n bits that remain unchanged out of 32-bit values, while the remaining 32-n bits are altered. Clearly, the Hamming distance is neither the larger the better nor the smaller the better; when n takes the value of 16, the randomness is optimal. In practice, the Hamming distance is not fixed, so it is desirable for most of its values to be distributed around 16. When the distribution of Hamming distances closely follows a normal distribution with a mean of 16, this indicates that the next key has as many possible values as possible, and thus its randomness is better.

For the DSAVK, AVKMSB, and chain key models, which have Hamming distance distributions relatively consistent with a normal distribution, as shown in Figures 4, it can be seen that although DSAVK and AVKMSB generally follow a normal distribution trend, they still exhibit significant fluctuations at certain special values. In contrast, the chain key model fits the normal distribution more closely.

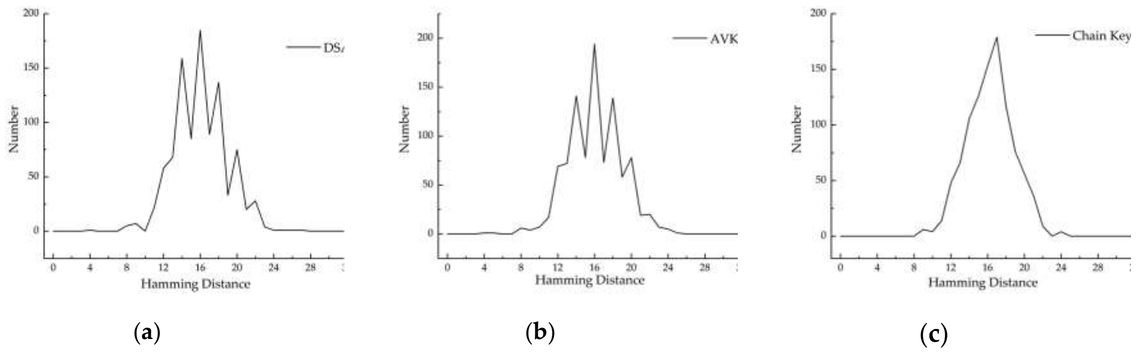


Figure 4. (a) DSAVK Hamming Distance Distribution; **(b)** MSBAVK Hamming Distance Distribution; **(c)** Hamming Distance Distribution of Chain key Model.

To provide a more formal representation, the randomness of the three schemes is quantified using the formulas for mean μ and variance σ^2 from probability statistics, as shown in equations (9) and (10).

$$\mu = \frac{\sum_{i=1}^{N} Hamming_i}{N} \tag{9}$$

$$\sigma^2 = \frac{\sum_{i=1}^{N} (Hamming_i - \mu)^2}{N} \tag{10}$$

The results obtained under these circumstances are shown in Table 7.

Table 7. Mean and Variance of the Three Schemes.

	Mean（μ）	Variance（σ^2）
DSAVK	15.962	9.394
MSBAVK	16.012	8.614
Chain key	16.267	6.407

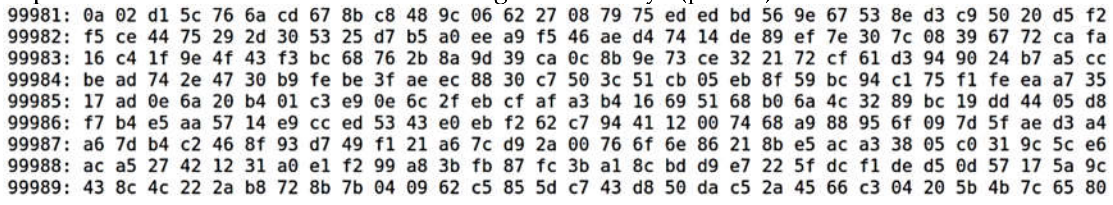
From the table above, it can be observed that while the means are relatively close, the chain key model exhibits a smaller variance. This indicates that in this experiment, the Hamming distances are mostly concentrated around 16.267, with less dispersion. However, for the other two schemes, the variance is comparatively larger than that of the chain key model. This suggests that under the specific conditions of this message set, the keys generated by these schemes are more predictable.

(2) Autocorrelation Coefficient

The autocorrelation coefficient [21] is a statistical measure that assesses the correlation between values in a sequence, with a range of [−1,1]. A value close to -1 indicates a significant negative correlation, meaning that an increase in one value is typically accompanied by a decrease in another; a value close to 0 suggests little to no correlation, indicating that changes are independent; a value close to 1 indicates a strong positive correlation, with consistent trends. Therefore, the autocorrelation coefficient can effectively analyze the association between node values in a hash chain.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 January 2025 doi:10.20944/preprints202501.2021.v1

To illustrate the randomness of the keys generated by this scheme, keys of 256-bit length were produced using the model, with a key generation cycle equivalent to an IPsec period (1 minute). Within this cycle, keys used for data encryption are analyzed. Due to the lengthy 256-bit length being unfavorable for analysis, it is divided into 32 sets of 8-bit data. The autocorrelation is calculated for each set of 8-bit data, resulting in 32 sets of autocorrelation coefficients. As shown in Figure 5, it depicts the hash chain node values of the generated keys (partial).



```
99981: 0a 02 d1 5c 76 6a cd 67 8b c8 48 9c 06 62 27 08 79 75 ed ed bd 56 9e 67 53 8e d3 c9 50 20 d5 f2
99982: f5 ce 44 75 29 2d 30 53 25 d7 b5 a0 ee a9 f5 46 ae d4 74 14 de 89 ef 7e 30 7c 08 39 67 72 ca fa
99983: 16 c4 1f 9e 4f 43 f3 bc 68 76 2b 8a 9d 39 ca 0c 8b 9e 73 ce 32 21 72 cf 61 d3 94 90 24 b7 a5 cc
99984: be ad 74 2e 47 30 b9 fe be 3f ae ec 88 30 c7 50 3c 51 cb 05 eb 8f 59 bc 94 c1 75 f1 fe ea a7 35
99985: 17 ad 0e 6a 20 b4 01 c3 e9 0e 6c 2f eb cf af a3 b4 16 69 51 68 b0 6a 4c 32 89 bc 19 dd 44 05 d8
99986: f7 b4 e5 aa 57 14 e9 cc ed 53 43 e0 eb f2 62 c7 94 41 12 00 74 68 a9 88 95 6f 09 7d 5f ae d3 a4
99987: a6 7d b4 c2 46 8f 93 d7 49 f1 21 a6 7c d9 2a 00 76 6f 6e 86 21 8b e5 ac a3 38 05 c0 31 9c 5c e6
99988: ac a5 27 42 12 31 a0 e1 f2 99 a8 3b fb 87 fc 3b a1 8c bd d9 e7 22 5f dc f1 de d5 0d 57 17 5a 9c
99989: 43 8c 4c 22 2a b8 72 8b 7b 04 09 62 c5 85 5d c7 43 d8 50 da c5 2a 45 66 c3 04 20 5b 4b 7c 65 80
```

Figure 5. Partial Key Hash Chain Node Values(partial).

By the definition of the autocorrelation coefficient as given in equation (11):

$$cor = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(x_{i+1} - \overline{x})}{\sum_{i=1}^{n}(x_i - \overline{x})^2} \tag{11}$$

We can calculate 32 sets of autocorrelation coefficients, which represent the correlation between each byte of the 256-bit key. The specific representation is shown in Figure 6.

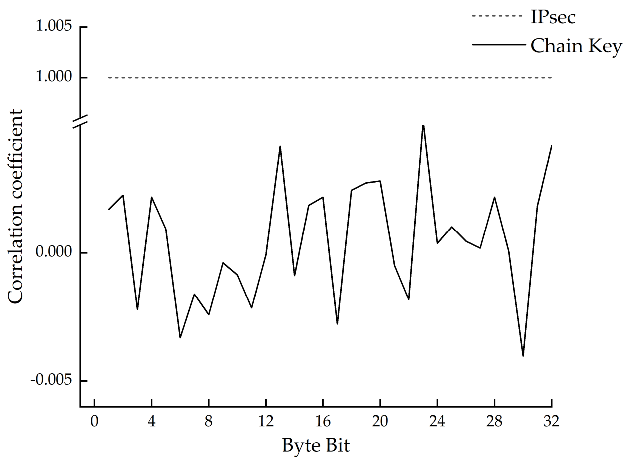


Figure 6. Autocorrelation Coefficient Analysis of Chain key and IPsec within a Cycle.

The results show that during a specific IPSec cycle, the correlation coefficient of the chained key sequence remains at a relatively constant level. This indicates that the correlation between any two corresponding bytes in the key sequence is very weak, with almost no significant linear relationship. Since IPSec did not update the keys during this period, the correlation coefficient consistently remained at 1.0.

Entropy

Entropy [22], similar to the autocorrelation coefficient, is a fundamental concept in randomness analysis used to quantify the uncertainty of a random variable. It is frequently employed to evaluate the degree of randomness in a sequence: a higher entropy value indicates a greater level of randomness, making it more challenging to predict subsequent states; conversely, a lower entropy value suggests that the random variable's values are more certain, thereby increasing predictability. For discrete sequences, such as those in the chain key model, assessing entropy is particularly crucial because it directly impacts the security of the key and the difficulty of cracking it. The definition of entropy in a discrete state is provided by equation (12).

$$H(X) = -\sum_i p(x_i) \log_2 p(x_i) \tag{12}$$

Within an IPsec cycle (1 minute), the keys used for data encryption are analyzed. Similarly, the 256-bit key is divided into 32 eight-bit bytes, and each byte is analyzed independently. The entropy results for each part are shown in Figure 7.

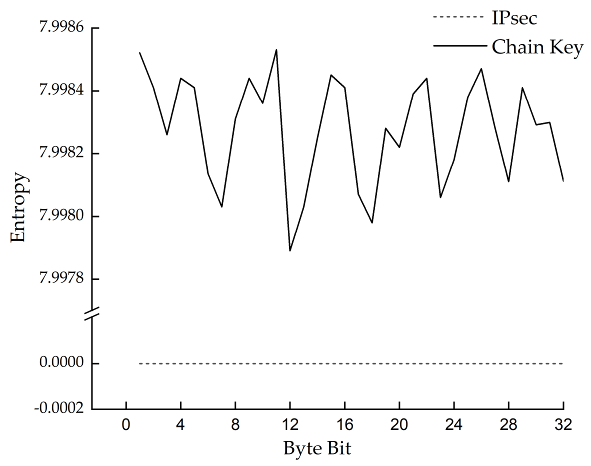


Figure 7. Entropy Value Analysis of Chain key and IPsec within a Cycle.

For uniformly distributed data with a value of entropy, the maximum entropy value $H_{\max} = \log_2(n) = \log_2(256) = 8$. As shown in the above figure 7, the entropy value is around 7.9984, which indicates that there is very little correlation between the values in the sequence, and the sequence tends to be highly random or uniformly distributed. In terms of randomness, this can be seen as a positive characteristic. Overall, the entropy value of the chain key indicates that this sequence behaves very close to uniform distribution and randomness in information theory. However, since IPSec does not update keys, its entropy value is 0, indicating that the key remains fixed throughout its lifecycle.

6.1. Analysis of Salsa Stream Cipher Based on Chain key

This experiment was conducted on Ubuntu 22.04 using the C programming language to test the efficiency of the $Salsa$, $hc128$ stream ciphers, and the AES block cipher in combination with the chain key algorithm. The analysis compares AES and IPsec, which utilizes AES, in terms of key update frequency and synchronization capabilities. Table 8 demonstrates that the chain key has a faster key update frequency compared to IPsec, which has a fixed interval for key updates and synchronization.

Table 8. Functionality of Various Models.

Algorithm	Encryption / Decryption	Authentication	Seed Key Update Frequency	Communication Key Error Correction and Synchronization
$hash-Salsa$	√	√	Fast (per-packet)	√
$Salsa$	√			
$hash-hc128$	√	√	Fast (per-packet)	√
$hc128$	√			
$IPSec$ (RSA)	√	√	Slow (fixed interval)	-
$IPSec$ (Pre-shared Key)	√	√	Slow (fixed interval)	-

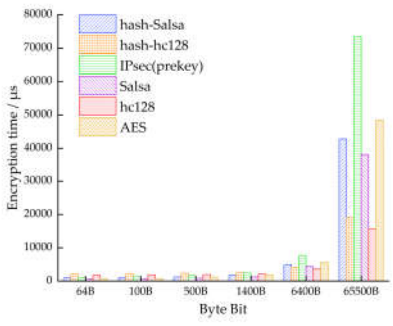
AES	√			
hash – *AES*	√	√	Fast (per-packet)	√

The tests were conducted on the encryption of 100 data packets with varying byte lengths of messages, where the encryption key was modified with each set of data packets. In contrast, the AES encryption key remained unchanged. The tests were divided into two aspects: with and without RSA authentication. The results are shown in Table 9.

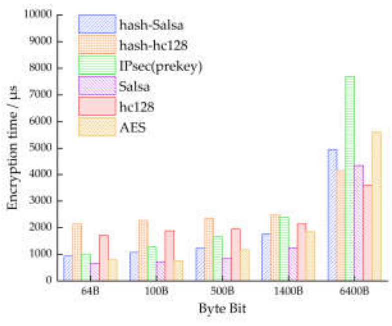
Table 9. Efficiency Comparison of Various Algorithms.

Test Content		64B	100B	500B	1400B	6400B	65500B
hash – *Salsa*	Encryption/μs	934	1071	1235	1761	4958	42804
	Decryption/μs	942	1123	1222	1824	5084	43660
Salsa	Encryption/μs	660	719	846	1247	4336	38156
	Decryption/μs	792	743	910	1326	4360	37870
hash – *hc*128	Encryption/μs	2138	2267	2347	2471	4146	19157
	Decryption/μs	2162	2296	2359	2500	4508	20177
*hc*128	Encryption/μs	1712	1881	1947	2138	3600	15636
	Decryption/μs	1774	1994	2039	2121	3378	16254
IPSec (Pre-shared Key)	Encryption/μs	1011	1277	1658	2387	7692	73484
	Decryption/μs	1043	1376	1660	2619	9028	85201
AES (No Authentication)	Encryption/μs	791	760	1167	2001	6545	60826
	Decryption/μs	819	848	1456	2349	7533	72613
IPSec(*RSA*)	Encryption/μs	8784	9156	9738	11999	17157	68877
	Decryption/μs	93298	96284	101692	103294	97948	161392

Due to the low efficiency of RSA compared to the pre-shared key scheme, its inclusion in a bar chart would affect the assessment of other algorithms. Therefore, the visualization bar charts for the other algorithms, excluding the RSA scheme, are shown in Figures 8 and 9.

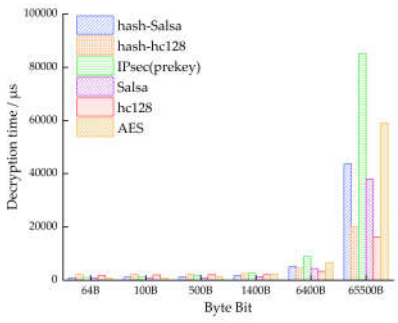


(**a**)

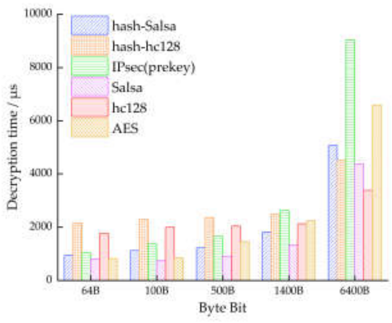


(**b**)

Figure 8. (**a**) The encryption time of various algorithms; (**b**) The encryption time of various algorithms in authentication conditions.



(**a**)



(**b**)

Figure 9. (**a**) The decryption time of various algorithms; (**b**) The decryption time of various algorithms in authentication conditions.

Compared to $hc128$, the $Salsa$ stream cipher is more suitable for packet-by-packet encryption in Ethernet environments where the number of bytes per packet is within 1400. After the introduction of packet authentication mechanisms, the operational efficiency of $hash-salsa$ is significantly better than that of IPsec. At 1400 bytes, the encryption efficiency increases by 35%, and the decryption efficiency increases by 43%. Even when the packet size exceeds a certain threshold (approximately 1000 bytes), the efficiency of the stream cipher with frequent key updates is higher than that of AES, which lacks authentication and does not update keys. In addition, if the authentication function of IPsec is RSA, the efficiency of RSA is not high, leading to significant additional overhead.

7. Conclusion

The hash-based chain key generation scheme proposed in this paper has the advantage of binding the randomness of plaintext and the irreversibility of hash algorithms to the keys. This minimizes the correlation between two different encryption keys, making it difficult for an attacker to crack any message, akin to a ciphertext-only attack. Additionally, by utilizing the message hash chain for key synchronization, it ensures that keys cannot be maliciously tampered with or destroyed by the communicating parties. Furthermore, unlike the concept of self-synchronizing stream ciphers, the chain key model requires the plaintext to generate the next key, which eliminates the possibility for attackers who can easily obtain ciphertext but find it difficult to decrypt it to bypass the plaintext and directly work with captured ciphertext, thus providing a certain level of resistance against related-key attacks. In the next phase, this model can be applied in various IoT environments to replace the complex IPsec for further research and development.

Author Contributions: Conceptualization, W.J.; Methodology, W.J. and J.D.; Software, Y.W; Formal analysis, W.J.; Writing – original draft, J.D. and Y.W.; Writing – review & editing, W.J., J.D.,Y.W. and H.D.; Project administration, W.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by National Key Research and Development Program of China, grant number 2022YFB2703000.

Data Availability Statement: The data presented in this study are available in this article. Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ghanem, K.; Ugwuanyi, S.; Hansawangkit; J.McPherson, R.; Khan, R.; Irvine, J. Security vs bandwidth: performance analysis between IPsec and OpenVPN in smart grid. In 2022 International Symposium on Networks, Computers and Communications (ISNCC), 2022,pp. 1-5. IEEE.
2. Shannon, C. E. Communication theory of secrecy systems[J]. The Bell system technical journal, 1949, 28(4): 656-715.
3. WANG,J.B.; ZHANG,W.Z. Provably-Secure Randomized Block Cipher Approaching to Perfect Secrecy[J]. Journal of Cryptologic Research,2021,8(5):808-819.
4. AI,X.; WU, M. D.; WU, X. D.; Design of one-time pad SM4 algorithm[J]. Cyberspace Security, 2018, 9(2): 20–23.
5. ZHANG, Y.J.; MA, J.M.; WEI, Y.Y. Self-synchronizing sequence cipher algorithm based on block encrypting synchronization information[J], Journal of Computer Applications, 2016, 36(S1): 42–45.
6. Prajapat, S.; Porwal, A.; Jaiswal, S.; Saifee, F.; Thakur, R. S. Generalized Parametric Model for AVK-Based Cryptosystem. In *Networking Communication and Data Knowledge Engineering:2018, Volume 2*,pp. 217-227. Springer Singapore.
7. Francq, J.; Besson, L.;Huynh, P.;Guillot, P.; Millerioux, G.; Minier, M. Non-triangular self-synchronizing stream ciphers. *IEEE Transactions on Computers*, 2020,*71*(1), 134-145.

8. LIU,H.F.; CAO,Y.M.; LIANG, X.L. Improved DES Based On Stream Cipher[J], Computer Applications and Software,2019,36(9). 317–320.
9. Chakrabarti P, Bhuyan B, Chowdhuri A, et al. A novel approach towards realizing optimum data transfer and Automatic Variable Key (AVK) in cryptography[J]. IJCSNS, 2008, 8(5): 241.
10. Goswami R S, Chakraborty S K, Bhunia A, et al. New techniques for generating of automatic variable key in achieving perfect security[J]. Journal of the Institution of Engineers (India): Series B, 2014, 95: 197-201.
11. Goswami R S, Chakraborty S K, Bhunia A, et al. New approach towards generation of automatic variable key to achieve perfect security[C]//2013 10th International Conference on information technology: new generations. IEEE, 2013: 489-491.
12. Singh B K, Banerjee S, Dutta M P, et al. Generation of automatic variable key to make secure communication[C]//Proceedings of the International Conference on Recent Cognizance in Wireless Communication & Image Processing: ICRCWIP-2014. Springer India, 2016: 317-323.
13. Chunka C, Banerjee S, Goswami R S. A Novel Key Generating Scheme of Automatic Variable Key[C]//2019 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS). IEEE, 2019: 1-5.
14. Jiang, W.; Wang, Y.; Ye, S. A Memorable Communication Method Based on Cryptographic Accumulator. *Electronics* 2024, *13*, 1081. https://doi.org/10.3390/electronics13061081HAN,M.X.;
15. JIANG,W.B.;GUO,Y.N. Signature and authentication method based on message hash chain[J]. Application Research of Computers. 2022, 39(4): 1183–1189.
16. Bernstein, D. J. The Salsa20 family of stream ciphers. In New stream cipher designs: the eSTREAM finalists. Berlin, Heidelberg: Springer Berlin Heidelberg. 2008, 84-97.
17. LI,Z.M. Design and analysis of the hash functions[D]. Beijing University of Posts and Telecommunications,2009.
18. WANG, J. Formalizaion of SHA256 algorithm for blockchain[D]. Beijing University of Chemical Technology,2022.
19. Goswami, R.S.;Chakraborty, S.K.; Bhunia, A.; Bhunia, C.T. New approach towards generation of automatic variable key to achieve perfect security. In *2013 10th International Conference on information technology: new generations* ,2013,pp. 489-491. IEEE.
20. Chunka, C.; Banerjee, S.; Goswami, R. S. A Novel Key Generating Scheme of Automatic Variable Key. In *2019 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)* ,2019,pp. 1-5. IEEE.
21. LI, Z.S. Pseudo-random sequence design and its randomness analysis[D]. University of Electronic Science and Technology of China,2019.
22. FAN, L.M.; FENG, D.G.; CHEN, H. Study on the Correlation Between Randomness Tests Based on Entropy[J]. Journal of Software, 2009, 20(7): 1967–1976.