electronics

MDPI

Article

Memory-Saving and High-Speed Privacy Amplification Algorithm Using LFSR-Based Hash Function for Key Generation

Enjian Bai *, Xue-qin Jiang and Yun Wu

School of Information Science & Technology, Donghua University, Shanghai 201620, China; xqjiang@dhu.edu.cn (X.-q.J.); wuyun_hit@dhu.edu.cn (Y.W.)
* Correspondence: baiej@dhu.edu.cn

Abstract: Privacy amplification is an indispensable procedure for key generation in the quantum key distribution system and the physical layer key distribution system. In this paper, we propose a high-speed privacy amplification algorithm that saves hardware memory and improves the key randomness performance. Based on optimizing the structure of the Toeplitz matrix generated by a linear feedback shift register, the core of our algorithm is a block-iterative structure hash function that is used to generate a secure key of arbitrary length. The proposed algorithm adopts multiple small Toeplitz matrices to compress the negotiation key for convenient implementation. The negotiated key is equally divided into multiple small blocks, and the multiplication operation of the negotiated key with the Toeplitz matrix is converted into a modular addition operation through an accumulator. The analysis results demonstrate that the algorithm has the advantages of saving memory and running quickly. In addition, the NIST randomness test and avalanche effect test on the key sequences indicate that the proposed algorithm has a favorable performance.

Keywords: quantum key distribution (QKD); physical layer security; privacy amplification (PA); Toeplitz matrix; randomness test

check for updates

Citation: Bai, E.; Jiang, X.-q.; Wu, Y. Memory-Saving and High-Speed Privacy Amplification Algorithm Using LFSR-Based Hash Function for Key Generation. *Electronics* **2022**, *11*, 377. https://doi.org/10.3390/electronics11030377

Received: 15 December 2021
Accepted: 24 January 2022
Published: 27 January 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Privacy amplification (PA) plays a vital role in the quantum key distribution (QKD) system and physical layer key distribution (PLKD) system. Both parties of legal communication, Alice and Bob, can apply the same PA algorithm to map the negotiated key into a shorter unconditional secure secret key in order to delete the information leaked to the eavesdropper Eve. At this time, the key information generated by Eve is almost zero; therefore, Alice and Bob can obtain the same secure unconditional key [1–5].

The main function of PA is to remove the leaked information from the negotiated key containing the amount of leaked information and to compress it into an absolutely secure final shared key. The earliest PA technology was proposed by Bennett et al. [6], where they proved that the technology can be applied to the quantum key distribution system to achieve unconditional safety [7]. In 2005, Renner et al. [8] proposed a general combination of security for PA that opened up a research direction on the combined security for later researchers. At present, PA is usually implemented using a universal hash function. Among them, the Toeplitz matrix is one of the widely used universal hash functions due to its simple structure and the fact that it can be implemented in parallel [9]. Alice and Bob multiply the negotiated key string X by the same Toeplitz matrix T to obtain another short key string X'. In general, since the length of X is much longer than X', the key string X is compressed, deleting the key information leaked to Eve [10]. The length of the negotiated key is usually very long, so the size of the Toeplitz matrix is particularly large; therefore, the requirements for the hardware resources will be higher, and even the calculation speed of the PA algorithm will be reduced.

To solve these problems, some more effective PA schemes have been proposed. In [11], the Toeplitz matrix was divided into multiple diamond-shaped blocks, and these diamond-shaped block operations were, respectively, operated in a field programmable gate array (FPGA) to improve the processing efficiency. However, it only considers the reconstruction of the Toeplitz matrix, and does not involve the adequate processing of the negotiated key with the Toeplitz matrix. In [12], a graphics processing unit (GPU) was used to speed up the implementation of the PA algorithm. When the GPU cannot be directly applied, the input data were divided into small batches and the fast Fourier transform (FFT) was used to accelerate the process. This algorithm requires numerous memory resources, since not only does the scheme not optimize the Toeplitz matrix structure, but when using FFT acceleration, the length of the negotiated key and the matrix must also be extended. In [13], the Toeplitz matrix was constructed by changing the continuous state of the linear feedback shift register (LFSR). At this point, only the first column elements of the matrix were stored, thus saving hardware storage resources. Furthermore, a PA algorithm is proposed, in which the question of whether the accumulator accumulates with the corresponding column of the Toeplitz matrix is determined according to the negotiated key bit value, and the secure key is obtained after multiple processes. Unfortunately, when the final secret key length is very long, the construction of the primitive polynomial involved in the algorithm is not easy. Moreover, the process of the negotiated key with the Toeplitz matrix in the algorithm is too simple to guarantee the performance of the algorithm. Tang et al. [14] filled both the negotiated key and the Toeplitz matrix with 0s and used FFT on the multi-core CPU to improve the computing power of the PA algorithm, but FFT also requires a lot of computing time. As far as we know, the focus of most existing PA schemes is on how to reduce the time taken by the PA algorithm to run. However, the randomness of the final secret key sequence obtained after applying the PA scheme is more important.

In this paper, we propose a high-speed PA algorithm that saves hardware memory and improves the key performance by optimizing the LFSR-based Toeplitz matrix. Our algorithm is an iterative hash function that can obtain any length of key sequence. Its core compression function is a very small LFSR-based Toeplitz matrix, where the row size of the matrix is 32, 64 or 128 and the required hardware storage is very small. At the same time, the compression process is a simple modular addition operation, and, thus, is very fast. The experimental results show that the randomness of the final key sequence has a favorable performance.

The rest of the paper is organized as follows. The Section 2 introduces some related principles of PA, the Section 3 introduces the implementation of the proposed algorithm, the Section 4 is the analysis of the experiment results and the Section 5 provides relevant discussion and the direction of further research. Finally, the Section 6 are presented.

2. Preliminary Work

2.1. Basic Principles of PA

PA is a process where the legitimate communication parties, Alice and Bob, use the same algorithm to extract the final secret key from the bit string obtained after the reconciliation and error correction process. After privacy amplification, Eve can hardly obtain any information about the key, and Alice and Bob use the key for secure communication [15].

The following gives the definition of PA from the perspective of information theory. Assume that, after reconciliation and error correction, Alice and Bob obtain the same binary string Y with length of n. At the same time, Eve obtains a random observation of Y, denoted by W, where W provides $t(t \leq n)$ bit information about Y, i.e., $H(Y|W) \geq n - t$. In order for Eve to obtain as little bit information about Y as possible, i.e., the value of t approaches zero, Alice and Bob choose a compression function $g : \{0,1\}^n \rightarrow \{0,1\}^r$. After $Z = g(Y)$, Eve has little information about Y [16].

If a universe class of hash functions is selected as the compression function g, we can calculate the amount of information of the negotiated key bit string leaked to Eve:

$$I(Z:g,W)=\frac{2^{-s}}{\ln 2}, \tag{1}$$

where $s=n-r-t$ denotes the security coefficient of PA [17].

2.2. Universal Hash Function Based on Toeplitz Matrix

Applied to the PA algorithm, the universe class of hash function H must meet the following requirements: (1) when the input is variable, the length of the output is fixed; (2) the calculation process is simple; (3) irreversible; (4) for a given x, finding $H(x)=H(y)$ and $x\neq y$ is computationally infeasible; (5) finding any pairs (x,y) that satisfy $H(x)=H(y)$ is computationally infeasible [18]. A hash function is randomly selected from the universal hash family, and a secure key is extracted from the negotiated key under the condition of a low collision probability. The collision probability means that even if Eve does not obtain the information between Alice and Bob, they can still obtain the same key. Therefore, it is very important to choose a universal hash function with a low collision probability for the PA algorithm. As one of the universal classes of hash functions, the Toeplitz matrix universal hash function has a simple structure and can be calculated in parallel, meeting all of the requirements of PA.

Since the elements that belong to the same diagonal are equal, the Toeplitz matrix is also called a diagonal constant matrix. In particular,

$$T=\begin{bmatrix} t_{0,0} & t_{0,1} & \cdots & t_{0,n-2} & t_{0,n-1} \\ t_{1,0} & t_{1,1} & \cdots & t_{1,n-2} & t_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ t_{m-2,0} & t_{m-2,1} & \cdots & t_{m-2,n-2} & t_{m-2,n-1} \\ t_{m-1,0} & t_{m-1,1} & \cdots & t_{m-1,n-2} & t_{m-1,n-1} \end{bmatrix} \tag{2}$$

is a Toeplitz matrix. For $\forall i,j,\theta\in N$ and $0\leq i+\theta\leq m-1$, $0\leq j+\theta\leq n-1$, $t_{i+1,j+1}=t_{i,j}$. It can be easily found from (2) that the Toeplitz matrix can be directly determined by the elements in the first row and the first column [19].

Define the modified Toeplitz matrix [14]

$$G(T')=(I_r|T')=\begin{bmatrix} 1 & 0 & \ldots & 0 & 0 & t_{0,0} & t_{0,1} & \cdots & t_{0,n-r-2} & t_{0,n-r-1} \\ 0 & 1 & \ldots & 0 & 0 & t_{1,0} & t_{1,1} & \cdots & t_{1,n-r-2} & t_{1,n-r-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & 0 & t_{r-2,0} & t_{r-2,1} & \cdots & t_{r-2,n-r-2} & t_{r-2,n-r-1} \\ 0 & 0 & \ldots & 0 & 1 & t_{r-1,0} & t_{r-1,1} & \cdots & t_{r-1,n-r-2} & t_{r-1,n-r-1} \end{bmatrix}, \tag{3}$$

where $r<n$, I_r is a $r\times r$ identity matrix and T' is a $r\times(n-r)$ Toeplitz matrix. The n length negotiated key W is divided into two parts, $W_r=(w_0,w_1,\cdots,w_{r-1})$ and $W_{T'}=(w_r,w_{r+1},\cdots,w_n)$. W_r^T and $W_{T'}^T$ are their corresponding transpose matrix. Then, the secure shared key K_{se} is obtained from the Toeplitz matrix universal hash function

$$K_{se}=G(T')W=I_r\times W_r^T\oplus T'\times W_{T'}^T=W_r^T\oplus T'\times W_{T'}^T. \tag{4}$$

The hash function (4) compresses the negotiated key with length n into a secure key with length r. Obviously, at least $n-1$ elements need to be stored, and each compression process needs to perform $r(n-r)$ multiplications and additions. In order to reduce the storage requirements and computational complexity, we use a LFSR to construct the Toeplitz matrix and hash function.

2.3. LFSR-Based Toeplitz Matrix and Hash Function

The linear feedback shift register is a mechanism that can generate a binary bit sequence. Its working principle is simply summarized: given the output of the previous state,

the linear function of the output is used as the input again, and this cycle is performed. The XOR function is often used as the single-bit linear function [20].

To construct a LFSR-based Toeplitz matrix, the following conditions must be met. The number of LFSR registers should be the same as the number of rows of the Toeplitz matrix, and the current value of the register represents the LFSR state. In the Toeplitz matrix $T_{m\times n}$, each column is a continuous LFSR state of length m. The number of matrix columns n are the total number of LFSR states. Therefore, the LFSR-based Toeplitz matrix is constructed as follows: (1) firstly, initialize the first column of the matrix, that is, determine the initial state of the LFSR; (2) then, move each column of the Toeplitz matrix down one unit, i.e., the LFSR moves one unit to the right; (3) next, update the first element of the current column by adding all XOR values obtained by XORing the elements of the previous column and the corresponding elements of the feedback polynomial; (4) finally, repeat (2) and (3) until all elements in the last column of the matrix are determined.

The elements of the feedback polynomial of degree m are XORed with the corresponding position elements of the previous column, and then the sum of all XOR values obtained is determined to the top elements of all columns except the first column. Suppose $p(x) = p_{m-1}x^m + p_{m-2}x^{m-1} + \cdots + p_0x + 1$ is the feedback polynomial of degree m, and the corresponding coefficients, except the constant coefficient, are $p = (p_0, p_1, \ldots, p_{m-1})$, and the current status of LFSR is $t_0 = (t_{0,0}, t_{1,0}, \ldots, t_{m-1,0})$. The top element of the jth-LFSR state is

$$t_{0,j} = \bigoplus_{i=0}^{m-1} t_{i,j-1} \cdot p_i = t_{j-1} \cdot p^T, j = 1, 2, \cdots . \tag{5}$$

Once the top element of all columns are determined, the entire Toeplitz can be determined [21,22]. Figure 1 shows the generation process of the LFSR-based Toeplitz matrix. When the feedback polynomial of LFSR $p(x)$ is a primitive polynomial, its output is a m-sequence. The m-sequence has similar statistical characteristics to the true random sequence, such as uniformity, run length and autocorrelation characteristics [23]. Therefore, the primitive polynomial will be selected as the feedback polynomial of LFSR, and then the LFSR-based hash function will be constructed according to the following method.

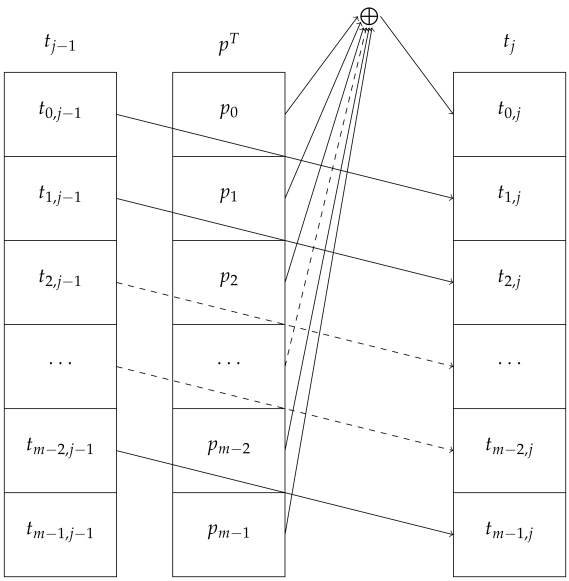

Figure 1. LFSR-based Toeplitz matrix generation process.

Now, let T' be the Toeplitz matrix generated from a LFSR with a primitive polynomial. The secure shared key K_{se} is obtained from the LFSR-based hash function

$$K_{se} = G(T')W = W_r \oplus W_{T'}T' = W_r \oplus \left(\oplus_{j=r,w_j=1}^{n} t_{j-r}\right). \quad (6)$$

The LFSR-based hash function (6) only needs to store r elements, whereas $n-1$ elements need to be stored in (4). Therefore, it saves hardware resources. Each compression process needs to perform, at most, $r(2n-2r+1)$ additions, and the computational complexity is also reduced.

3. Proposed Algorithm

Most existing PA schemes directly process the negotiated key using the Toeplitz matrix, and use some hardware-based methods to speed up the process. However, these methods do not consider how to improve the randomness of the key sequence. The LFSR-based hash function saves a certain amount of hardware storage resources and improves the processing efficiency of the algorithm. Therefore, we propose a high-speed PA algorithm that saves hardware memory and improves the randomness performance.

Table 1 gives some notations involved in the algorithm. Figure 2 depicts the overall processing of the proposed PA algorithm.

Table 1. Notations.

Notation	Definition
W	Negotiated key
n	Length of negotiated key
S_{ke}	Final secret key
r	Length of final secret key
W_r	The first r bits of W
$W_{T'}$	The last $n-r$ bits of W
$W_{T'j}$	The jth sub-block of $W_{T'}$
k	Length of sub-blocks
N	Number of sub-blocks
m	Number of LFSR registers
F	LFSR-based compression module
H_i	ith m-bit buffer used to hold intermediate result of hash function
IV	m-bit initial value of LFSR
C_r	The last r bits of $H_1 \| \| H_2 \| \| \cdots \| \| H_N$

The length of the negotiated key W is n, which is divided into $W_r = (w_0, w_1, \cdots, w_{r-1})$ and $W_{T'} = (w_r, w_{r+1}, \cdots, w_n)$. The length of the final secure shared key K_{se} is r. For convenience, $W_{T'}$ is further divided into several small sub-blocks with length k, and the size of the LFSR-based Toeplitz matrix is $m \times k$, where $m \leq k$. The core idea is to use an m-order LFSR to construct a Toeplitz matrix of size $m \times k$ to achieve key compression. The last $n-r$ bits of the negotiated key $W_{T'}$ are divided into N sub-blocks of length k, and the last blocks may be filled with $kN-n+r$ zeros. The LFSR state is initalized using IV to generate the Toeplitz matrix, and the first sub-block is processed in the compression function F. The accumulation form is used to improve the calculation ability of the PA algorithm, and the intermediate result H_1 is obtained. Then, H_1 is used as the initial state of the LFSR for the next sub-block, and this is repeated until all sub-blocks are processed. All intermediate keys are combined and the last r bits and XOR are taken with W_r to obtain the final secret key. The specific process of the algorithm is described in detail below.

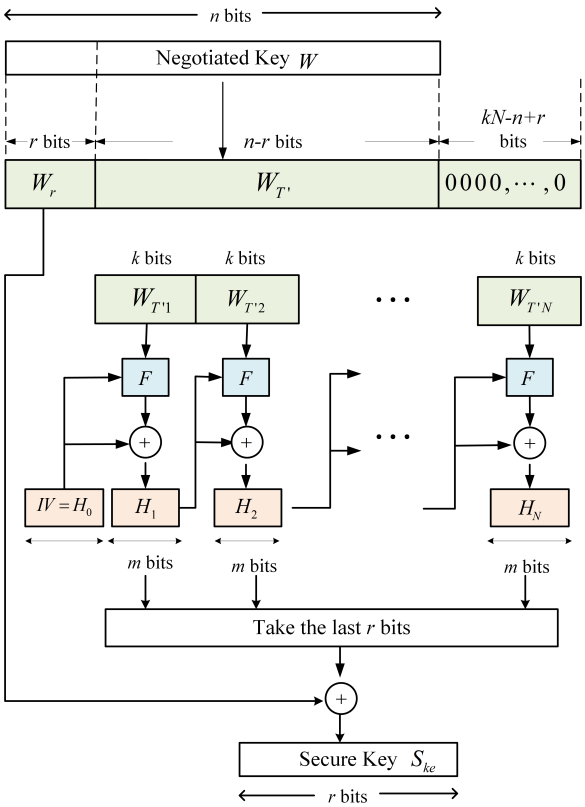

Figure 2. Secure key generation using our PA algorithm.

Step 1: Append padding bits. For given parameters k, m, n and r, the negotiated key W is divided into W_r and $W_{T'}$, of which, the lengths are r and $n-r$, respectively. Padding multiple zeros so that $W_{T'}||0\cdots0$ can be divided into N sub-blocks with length k, $N = \lceil \frac{n-r}{k} \rceil$ is satisfied. In Figure 2, the expanded negotiated key is represented as $W_r||W_{T'1}||W_{T'2}||\cdots||W_{T'N}$;

Step 2: H_0 initialization. Multiple m-bit buffers, $H_0, H_1, \cdots, H_N$, are used to hold intermediate results of the PA algorithm. The specific method of initializing registers H_0 is to conduct a module two operation through the remainder sequence obtained by the square root of the prime number 2, and then to take the first m bits of the binary sequence. For instance, when $m = 32$, H_0 is set to C8C7F0A9. The value of H_0 at this time is the initial state of the LFSR, denoted as $\{h_0[0], h_0[1], \cdots, h_0[m-1]\}$. H_i is the intermediate compression result of sub-block $W_{T'i}$ and is used as the initial state of LFSR when processing the next sub-block $W_{T'(i+1)}$;

Step 3: Process sub-block $W_{T'i}$. The heart of our PA algorithm is the LFSR-based hash fuction F in Figure 2. The logic principle of F is illustrated in Figure 3. The negotiated sub-block key $W_{T'i} = \{w_{T'i}[0], w_{T'i}[1], \cdots, w_{T'i}[k-1]\}$ and $H_{i-1} = \{h_{i-1}[0], h_{i-1}[1], \cdots, h_{i-1}[m-1]\}$ are input into F, and a Toeplitz matrix of size $m \times k$ is constructed according to the initial state H_{i-1} of the LFSR. When $w_{T'i}[j] = 1 (j = 0, 1, \cdots, k-1)$, the jth state of the Toeplitz matrix is accumulated with the corresponding value in the accumulator. When $w_{T'i}[j] = 0 (j = 0, 1, \cdots, k-1)$, no processing is carried out. If this is regarded as a loop, k loops are required. Then, the value in the accumulator is XORed with the value in the previous buffer H_{i-1}. At this time, the value in the accumulator is the intermediate key H_i with a length of m bits;

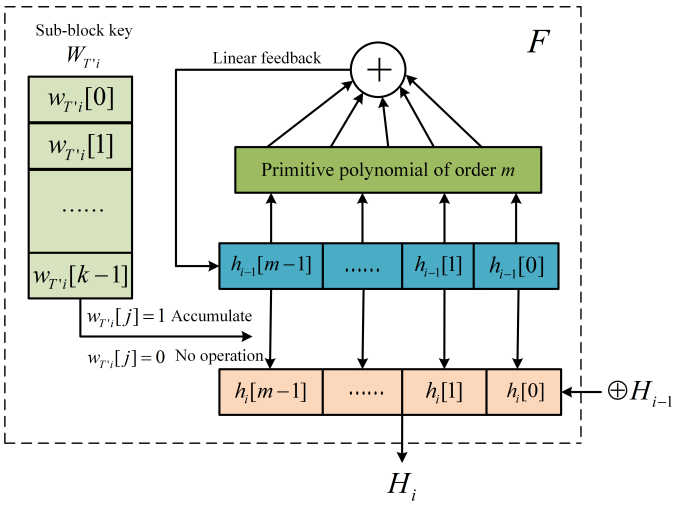

Figure 3. The operation of compression module F.

Step 4: Output. After all N k-bit sub-blocks of the negotiated key have been processed, all intermediate results are merged and denoted as $H_1, H_2, \cdots, H_N$, and the last r bits are XORed with W_r.Then, the final secure shared key K_{se} is obtained.

We summarize the proposed PA algorithm as follows:

$$H_0 = IV, \tag{7}$$
$$H_i = \oplus(H_{i-1}, F(H_{i-1}, W_{T'i})), \tag{8}$$
$$C_r = \text{the last } r \text{ bits of } H_1||H_2||\cdots||H_N, \tag{9}$$
$$S_{ke} = W_r \oplus C_r. \tag{10}$$

Compared with the algorithm in [13], we divide the large Toeplitz matrix with r rows and n columns into multiple small Toeplitz matrices with m rows and k columns, and the primitive polynomials of LFSR can be easily found. Compared with the algorithm proposed by Tang et al. [14], our PA scheme converts the processing of the negotiated key with the Toeplitz matrix into an accumulated form, which is more simple than the FFT method. At the same time, each intermediate key is used as the initial state of the LFSR for processing the next negotiated key block, so as to improve the randomness of the final secret key.

4. Results

The computing power necessary to deal with the negotiated key with the Toeplitz matrix, as well as the randomness of generating the final secure shared key, are the main performance indicators for evaluating a PA algorithm. Generally, the higher the data processing rate, the less time it takes to process the negotiated key and Toeplitz matrix, and the stronger the computing power of the PA algorithm. The higher the randomness, the better the performance of generating the final secure shared key.

We have carried out software implementation and analysis to highlight the advantages of the proposed algorithm. A simulation experiment is performed using Python language. The computer is Intel(R) i5-10210U, CPU 2.40 GHz and memory 16 GB, and the operating system is Microsoft Windows 10.

4.1. Memory Analysis

When using an ordinary Toeplitz matrix in PA, there is a need to store $r + n - 1$ elements for generating a r bit secret key from the n bit negotiated key. Using the method in [13], the Toeplitz matrix is generated by a LFSR, and there is only a need to store r bit LFSR states and r bit accumulators, and thus $2r$ bits in total. However, when r is large, it is not easy to find the primitive polynomial of order r. In our PA algorithm, the

negotiated sub-block key and the initial values in the first buffer H_0 will be processed as an input to module F. The operations of generating Toeplitz matrices and processing the negotiated sub-block key with the corresponding Toeplitz matrix are all performed in real time. Accordingly, we only need to store m elements in H_0, m elements in LFSR states and r bits W_r, r bits C_r, $2(r+m)$ bits in all. This memory requirement is also less than $2(r+k)$ [14] when $m < k$. Parameter m is suggested to be 32, 64 or 128, which means that the order of the LFSR primitive polynomial is very small. For the millions of negotiated keys in the quantum key distribution system, the storage space is greatly saved.

4.2. Computational Complexity Analysis

Since the generation of the Toeplitz matrix and the processing of the negotiated key in the PA algorithm are binary bit operations, we can derive the computational complexity of our PA algorithm and compare it to the algorithm in [13,14]. We assume that the negotiated key length is $n = r + k \times N$ bits and that the final key length is r bits.

According to [13], r elements were initialized first. Then, based on the LFSR construction method as described in Section 2.3, the Toeplitz matrix was directly generated according to the selected primitive polynomial. At this point, an r bit operation will be performed $n-1$ times. Finally, the r bit accumulator was used to deal with the negotiated key and Toeplitz matrix directly. At this time, the r bit operation will be performed n times. Therefore, the total number of binary bit operations is calculated as $2nr - r$.

In our algorithm, we first divide the negotiated key $W_{T'}$ into N blocks, and each sub-block negotiated key is processed separately, which requires N periods. Then, each sub-block is input into module F with the initial values of the m-order LFSR. At this time, there is a need for an m bit operation to be carried out $k-1$ times to generate the Toeplitz matrix, an m bit operation to be carried out k times to accumulate and an m bit operation to be carried out to produce the intermediate results $H_i, i = 1, 2, \cdots, N$ in one period. Finally an r bit operation is performed to obtain the final secret key. Therefore, we only need to calculate $2mkN + r$ binary bit operations. Compared with the fast-Fourier-transform-enhanced high-speed PA scheme in [14], which needs at least $2(k-1)kN + r(k+1)$ binary operations, our PA algorithm has a low computational complexity.

In order to visually compare, we plot the computational complexity of these three PA algorithms in Figures 4 and 5. Figure 4 shows the amount of binary operations when r changes from 150, 250, 500, 1000, 2500 and 4000 to 5000 for a fixed n = 10,000. It can be seen that the growth of [13] is linear, whereas [14] and our algorithm decrease with the increase in r. Obviously, the latter two algorithms are much better for a fixed r. At the same time, the proposed algorithm is better than [14], and the amount of binary operations has a faster decline speed. Figure 5 shows the amount of binary operations when n changes from 1250, 2500, 25,000, 125,000, 250,000 and 1,250,000 to 2,500,000 for a fixed r = 250. It can be seen that [13] increases rapidly, whereas [14] and our algorithm increase slowly with the increase in n. The proposed algorithm has the slowest growth in the amount of binary operations among the three algorithms.

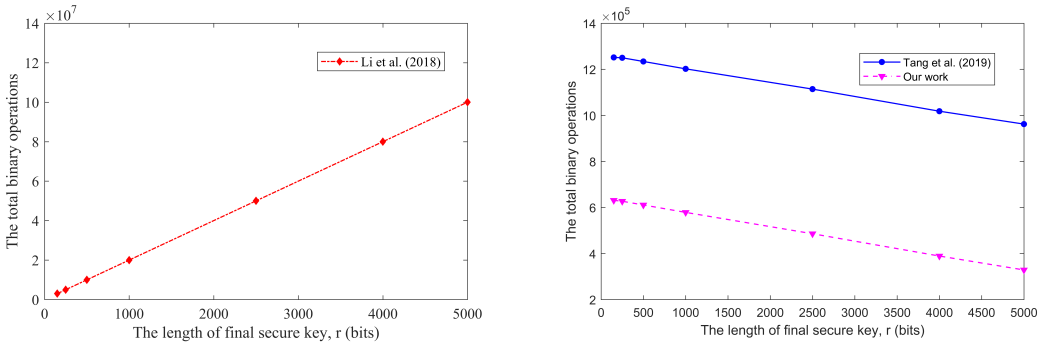

Figure 4. The total binary operations of [13,14] and proposed algorithm when k = 64, m = 32, n = 10,000 and r is changing from 150, 250, 500, 1000, 2500 and 4000 to 5000.

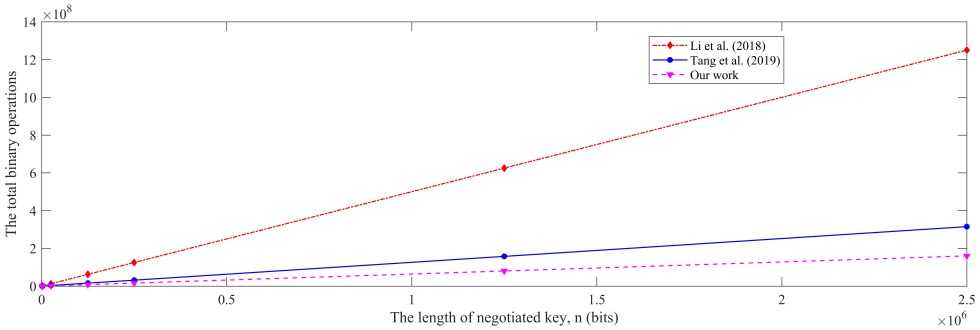

Figure 5. The total binary operations of [13,14] and proposed algorithm when $k = 64$, $m = 32$, $r = 250$ and n is changing from 1250, 2500, 25,000, 125,000, 250,000 and 1,250,000 to 2,500,000.

4.3. Running Speed Analysis

For the PA scheme, we should pay attention to its running time. It can be seen from the above analysis that the scheme of [13] has no advantages in terms of storage and calculation. Therefore, the following analysis will mainly compare our work with the scheme of [14].

First, we set the negotiated key length to 10,000 bits, the compression ratio r/n to 10% and the sub-block length k to 128 bits. Using different primitive polynomials of LFSR, the data calculation ability is analyzed. When the order of the primitive polynomial is 32, 64 and 128, the corresponding processing rate of our algorithm is 18.87 kbps, 7.58 kbps and 4.05 kbps, respectively. Obviously, the higher the order of the primitive polynomial, the lower the ability to process data. The reason is that more data are stored in the register, and it takes more time to process the data.

Then, we observe the influence of the sub-block size and compression ratio on the data processing rate. Table 2 compares the processing rate of Tang et al. [14] and the proposed scheme. Where the negotiated key length is 1M bits, the compression ratio is 10%, 20% and 40%, (that is, the final secret shared key length is 0.1 M, 0.2 M, 0.4 M bits), the sub-block length is 32 bits, 64 bits and 128 bits, respectively, and the LFSR order in our algorithm is 32.

It can be obtained from Table 2 that, with the increase in the sub-block length, the computing power of the FFT processing negotiated key with the Toeplitz matrix is gradually decreasing, but our solution changes very little in the data processing rate. The reason is that the longer the block, the more data that need to be FFT multiplied [14], but our scheme adopts the cumulative form, which is not sensitive to the length of the block, so there is almost no change in the processing rate. On the other hand, the change in the compression ratio has a slight impact on the processing capacity of the PA algorithm because the larger the compression ratio, the smaller the $W_{T'}$ and the less the processed data, so the time will be reduced and the rate will be higher using our scheme. When the block is 32 bits, the algorithm [14] is close to the rate of our solution for processing the negotiated key and Toeplitz matrix, so the key block using the FFT method should be as small as possible.

Table 2. The data processing rate of different sub-blocks when the compression ratio is 10%.

Compression Ration r/n	**Sub-Block Length** k **(bits)**	**Tang et al. [14] (kbps)**	**Our Scheme (kbps)**
10%	32	16.16	17.54
	64	13.21	18.12
	128	9.24	18.4
20%	32	16.69	18.11
	64	13.82	18.5
	128	9.41	19.03
40%	32	17.74	19.44
	64	14.29	19.54
	128	9.64	19.59

Figure 6 shows the time required to process data in the PA algorithm using FFT [14] and our scheme when the length of the negotiated key is from 1 M to 3 M bits, the block length is 64 and the compression ratio is 20%. It can be seen that when the negotiated key length is 3 M, the accumulation method for constructing the Toeplitz matrix based on LFSR is almost three-fifths of the FFT method used by Tang et al. [14]. When the same amount of data is input, the rate at which our scheme processes the negotiated key is better than the algorithm proposed by Tang et al. [14].

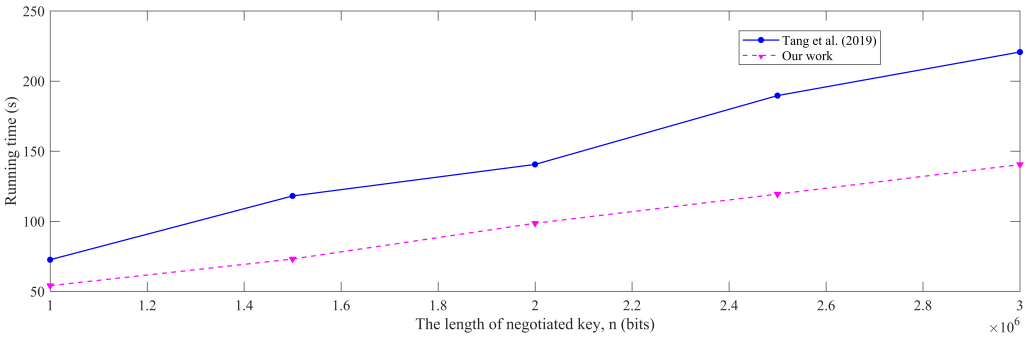

Figure 6. The running time using FFT [14] and our scheme when $k = 64$, $m = 32$ and n is changing from 1 M, 1.5 M, 2 M and 2.5 M to 3 M bits.

4.4. Performance Analysis

4.4.1. Key Randomness Analysis

It is critical to ensure the randomness of the binary sequence obtained by applying the PA algorithm, since it determines the performance of the entire key distribution system [24]. The NIST test suite is applied to test the randomness of the binary bit sequence obtained by [14] and our algorithm. This suite includes multiple tests, each of which returning p-values. When $p \leq 0.01$, it indicates that the binary bit sequence has not passed the corresponding test, and when $0.01 < p \leq 1$, the binary sequence has passed the corresponding test. The higher the p-value, the better the randomness of this sequence [25].

We set the length of the negotiation key to 1M bits, the compression ratio to 10% and the block length to 128 bits, and observe the effect of the LFSR order on randomness. Table 3 shows the randomness of our algorithm using 32-, 64- and 128-order primitive polynomials to generate the Toeplitz matrix. It can be seen that NIST-related tests have been passed in all cases.

Table 3. The randomness of different primitive polynomials when the compression ratio is 10%.

Dataset	**The Order of LFSR m (bits)**		
	32	**64**	**128**
Serial	0.580564	0.600267	0.498089
Runs	0.519745	0.576610	0.577771
Random Excursions Variant	0.464681	0.533625	0.602457
Random Excursions	0.509029	0.554956	0.621374
Non Overlapping Template Matching	0.858324	0.880256	0.903727
Monobit	0.586127	0.651421	0.551028
Maurers Univeral	0.985809	0.986209	0.984463
Longest Run Ones in a Block	0.600623	0.610743	0.606504
Linear Complexity	0.590901	0.639073	0.517903
Frequency within Block	0.568941	0.587819	0.514995
FFT	0.503946	0.669390	0.437223
Cumulative Sums	0.571402	0.637480	0.540185
Approximate Entropy	0.588679	0.601484	0.478181

Table 4 compares the randomness of the algorithm proposed by Tang et al. [14] and our algorithm. The block lengths are 32 bits and 64 bits. Through data processing rate analysis (Table 2), when the block length is 32 bits, the computing power of the two algorithms is similar. However, it can be clearly seen from the experimental results in Table 4 that when the block length is 32 bits, the randomness of [14] is worse than ours. The reason is that our scheme adopts an iterative compression structure. The processing result is accumulated with the result in the previous round of registers instead of directly multiplying and outputting.

Table 4. The randomness comparison of two schemes when the compression ratio is 10%.

Dataset	Sub-Block Length 32 (bits)		Sub-Block Length 64 (bits)	
	[14]	Our Work	[14]	Our Work
Serial	0.476352	0.545938	0.504013	0.557471
Runs	0.502624	0.512228	0.512011	0.542043
Random Excursions Variant	0.491531	0.519558	0.560732	0.664592
Random Excursions	0.446167	0.475485	0.533430	0.538412
Non Overlapping Template Matching	0.904998	0.962631	0.980996	0.982631
Monobit	0.471014	0.547625	0.500762	0.563445
Maurers Univeral	0.968589	0.998544	0.998629	0.998488
Longest Run Ones in a Block	0.512048	0.542612	0.522744	0.522456
Linear Complexity	0.033841	0.045339	0.053053	0.058726
Frequency within Block	0.481607	0.509643	0.501623	0.529392
FFT	0.487356	0.514130	0.502167	0.590420
Cumulative Sums	0.473373	0.539399	0.494547	0.567423
Approximate Entropy	0.507883	0.525643	0.510313	0.536329
Binary Matrix Rank	0.500990	0.525636	0.563987	0.584178

4.4.2. Avalanche Effect Analysis

The avalanche effect, an ideal property of hash algorithms, is when the smallest change in the input (for example, inverting one binary bit) can also result in drastic changes in the output (half of the output bits invert). If a cryptographic function does not exhibit a certain degree of avalanche characteristics, it is considered to have poor randomization characteristics; therefore, the cryptanalyst can infer the input from the output [24].

We test the avalanche effect characteristics of our PA algorithm. We first applied the PA algorithm to generate the secret key with a length of 128 bits (generally, the key length of the block cipher is at least 128 bits, such as AES), then changed the negotiated key on a small scale, such as one bit, two bits and three bits (every time, the position of the changed bit is different), to compare the difference between the secret key obtained by using the changed negotiated key as the input of our PA algorithm with the previous secret key. From Table 5, we can clearly find that, each time we change the negotiated key, the final secret key sequence obtained by applying our PA scheme is nearly half changed. Therefore, our PA algorithm has avalanche effect characteristics and can be used as a good hash algorithm for key distribution.

Table 5. The avalanche effect of our PA algorithm (twelve times for each experiment).

Bit Changes in the Negotiation Key	Bit Changes in the Final Secret Key
1	64/62/61/65/67/66/63/62/69/62/61/68
2	63/62/67/66/63/61/65/64/68/61/62/69
3	69/61/63/64/66/68/65/65/64/62/68/63

5. Discussion

The solution in this paper is mainly to use the feature of the LFSR-based Toeplitz matrix that can save resources and implement quickly, and to use the advantages of the iterative structure to construct the hash function.

We test the performance of the proposed algorithm on the software platform. The results show that, under the same conditions, the order of LFSR has little effect on the randomness of the key, but it will have a great impact on the data processing speed. The smaller the value, the faster the processing speed. The block size basically does not affect the speed, randomness and storage space. Therefore, the order of LFSR can be selected as 32 and the sub-block size is set to 64. However, NIST randomness tests do not provide unconditional security, which is standard in QKD. In QKD, a family of hash functions for privacy amplification is required to be two-universal, or, at least, ϵ-almost two-universal (with some small ϵ). Since there is no proof that the presented algorithm generates a two-universal family of test functions, a major obstacle for the use of this algorithm in QKD and other physical unconditionally secure key distribution schemes is presented. It is very important to prove this property in future.

The function of the QKD or PLKD system is to distribute the key for classical cryptography. For block cipher systems, the length of the key is generally 128, 192, 256 or longer. For public key cryptosystems, the required key lengths are generally 1024, 2048 and 4096. In both cases, the parameter r in our algorithm is a relatively small value that can directly adopt the scheme in this paper. However, if it is directly used in a stream cipher, the key length required is very large, and the storage scale required by the scheme will also be very large. It can be combined with the classical stream cipher system to generate the seed key of the stream cipher algorithm.

In addition, the Toeplitz matrix can be implemented in FPGA [12] to further improve the processing efficiency, and another feasible scheme is to realize the acceleration effect by using GPU, such as in [12]. This will be studied in the future.

6. Conclusions

In this paper, we proposed a privacy amplification algorithm for key distribution. On the basis of optimizing the Toeplitz matrix through LFSR, the algorithm divided a large Toeplitz matrix into multiple small Toeplitz matrices, and then processed each small Toeplitz matrix with the corresponding negotiated key by an iterative approach to achieve the purpose of sufficient processing. The implementation results and analysis demonstrated that our algorithm can save hardware memory resources, improve the randomness performance and run at high speed.

Author Contributions: E.B., Conceptualization, methodology, software, writing—original draft preparation; X.-q.J., writing—review and editing, project administration, funding acquisition. Y.W., supervision, writing—review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Fundation of Shanghai (Grant no. 20ZR1400700), the Shanghai Municipal Science and Technology Major Project (Grant no. 2019SHZDZX01) and the State Key Laboratory of Advanced Optical Communication Systems and Networks (Grant no. 2020GZKF002), China.

Data Availability Statement: Data are available in a publicly accessible repository that does not issue DOIs. Publicly available datasets were analyzed in this study, and the data used to support the findings of this study are available from the corresponding author on reasonable request.

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

References

1. Gilbert, G.; Hamrick, M. Secrecy, Computational Loads and Rates in Practical Quantum Cryptography. *Algorithmica* **2002**, *34*, 314–339.
2. Zhang, J.Q.; Duong, T.Q.; Marshall, A.; Woods, R. Key Generation from Wireless Channels: A Review. *IEEE Access* **2017**, *4*, 614–626. [CrossRef]
3. Melki, R.; Noura, H.N.; Mansour, M.M.; Chehab, A. A Survey on OFDM Physical Layer Security. *Phys. Commun.* **2019**, *32*, 1–30. [CrossRef]

4. Mukherjee, A.; Fakoorian, S.A.A.; Huang, J.; Swindlehurst, A.L. Principles of Physical Layer Security in Multiuser Wireless Networks: A Survey. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 1550–1573. [CrossRef]
5. Bottarelli, M.; Epiphaniou, G.; Ben Ismail, D.K.; Karadimas, P.; Al-Khateeb, H. Physical Characteristics of Wireless Communication Channels for Secret Key Establishment: a Survey of the Research. *Comput. Secur.* **2018**, *78*, 454–476. [CrossRef]
6. Bennett, C.H.; Zekrifa D.M.S.; Robert, J.M. Privacy Amplification by Public Discussion. *Siam J. Comput.* **2012**, *17*, 210–229. [CrossRef]
7. Bennett, C.H.; Brassard, G.; Crepeau, C.; Maurer, U.M. Generalized Privacy Amplification. *IEEE Trans. Inf. Theory* **1995**, *41*, 1915–1923. [CrossRef]
8. Renner, R.; Konig, R. Universally Composable Privacy Amplification Against Quantum Adversaries. In Proceedings of the Second International Conference on Theory of Cryptography, Cambridge, MA, USA, 10–12 February 2005; Springer: Berlin, Germany, 2005; pp. 407–425.
9. Carter, J.L.; Wegman, M.N. Universal Classes of Hash Functions. *J. Comput. Syst. Sci.* **1979**, *18*, 143–154. [CrossRef]
10. Grosshans, F.; Assche, G.V.; Wenger, J.; Brouri, R.; Cerf, N.J.; Grangier, P. Quantum Key Distribution Using Gaussian-modulated Coherent States. *Nature* **2003**, *421*, 238–241. [CrossRef]
11. Yang, S.S.; Bai, Z.L.; Wang, X.Y.; Li, Y.M. FPGA-based Implementation of Size-adaptive Privacy Amplification in Quantum Key Distribution. *IEEE Photonics J.* **2017**, *9*, 7600308. [CrossRef]
12. Wang, X.Y.; Zhang, Y.C.; Yu, S.; Guo, H. High-speed Implementation of Length-compatible Privacy Amplification in Continuous-variable Quantum Key Distribution. *IEEE Photonics J.* **2018**, *10*, 7600309. [CrossRef]
13. Li, D.W.; Huang, P.; Zhou, Y.M.; Li, Y.; Zeng, G.H. Memory-saving Implementation of High-speed Privacy Amplification Algorithm for Continuous-variable Quantum Key Distribution. *IEEE Photonics J.* **2018**, *10*, 7600712. [CrossRef]
14. Tang, B.Y.; Liu, B.; Zhai Y.P.; Wu, C.Q.; Yu W.R. High-speed and Large-scale Privacy Amplification Scheme for Quantum Key Distribution. *Sci. Rep.* **2019**, *9*, 15733. [CrossRef] [PubMed]
15. Bennett, C.H.; Brassard, G. Quantum Cryptography: Public Key Distribution and Coin Tossing. *Theor. Comput. Sci.* **2014**, *560*, 7–11. [CrossRef]
16. Diamanti, E.; Leverrier, A. Distributing Secret Keys with Quantum Continuous Variables: Principle, Security and Implementations. *Entropy* **2015**, *17*, 6072–6092. [CrossRef]
17. Wegman, M.N.; Carter, J.L. New Hash Functions and Their Use in Authentication and Set Equality. *J. Comput. Syst. Sci.* **1981**, *22*, 265–279. [CrossRef]
18. Wang, J.D.; Zhang, T.; Song, J.K.; Sebe, N.; Shen, H.T. A Survey on Learning to Hash. *IEEE Trans. Pattern Anal. Mach. Intell.* **2018**, *40*, 769–790. [CrossRef] [PubMed]
19. Wax, M.; Kailath, T. Efficient Inversion of Toeplitz-block Toeplitz Matrix. *IEEE Trans. Acoust. Speech, Signal Process.* **1983**, *31*, 1218–1221. [CrossRef]
20. Peinado, A.; Fuster-Sabater, A. Generation of Pseudorandom Binary Sequences by Means of Linear Feedback Shift Registers (LFSRs) with Dynamic Feedback. *Math. Comput. Model.* **2013**, *57*, 2596–2604. [CrossRef]
21. Deepthi P.P.; Sathidevi, P.S. Design, Implementation and Analysis of Hardware Efficient Stream Ciphers Using LFSR-based Hash Functions. *Comput. Secur.* **2009**, *28*, 229–241. [CrossRef]
22. Cai, C.C.; Bai, E.J.; Jiang, X.Q.; Wu, Y. Simultaneous Audio Encryption and Compression Using Parallel Compressive Sensing and Modified Toeplitz Measurement Matrix. *Electronics* **2021**, *10*, 2902. [CrossRef]
23. Menezes, A.; Oorschot, P.V.; Vanstone, S. *Handbook of Applied Cryptography*; CRC Press: Boca Raton, FL, USA, 1997.
24. Luby, M. *Pseudorandomness and Cryptographic Applications*; Princeton University Press : Princeton, NJ, USA, 1996.
25. Bassham, L.; Rukhin, A.; Soto, J.; Nechvatal, J.; Smid, M.; Leigh, S.; Levenson, M.; Vangel, M.; Heckert, N.; Banks, D. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD. Available online: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=906762 (accessed on 10 December 2021).