Fast implementation of length-adaptive privacy amplification in quantum key distribution*

Zhang Chun-Mei(张春梅)[a], Li Mo(李 默)[a], Huang Jing-Zheng(黄靖正)[a], Patcharapong Treeviriyanupab[b], Li Hong-Wei(李宏伟)[a], Li Fang-Yi(李芳毅)[a], Wang Chuan(王 川)[a], Yin Zhen-Qiang(银振强)[a]†, Chen Wei(陈 巍)[a], Keattisak Sripimanwat[b], and Han Zhen-Fu(韩正甫)[a]

[a] *Key Laboratory of Quantum Information, Chinese Academy of Sciences, University of Science and Technology of China, Hefei* 230026, *China*
[b] *Optical and Quantum Communications (OQC) Laboratory, National Electronics and Computer Technology Center (NECTEC), National Science and Technology Development Agency (NSTDA), Thailand*

(Received 5 February 2014; revised manuscript received 11 March 2014; published online 23 July 2014)

Post-processing is indispensable in quantum key distribution (QKD), which is aimed at sharing secret keys between two distant parties. It mainly consists of key reconciliation and privacy amplification, which is used for sharing the same keys and for distilling unconditional secret keys. In this paper, we focus on speeding up the privacy amplification process by choosing a simple multiplicative universal class of hash functions. By constructing an optimal multiplication algorithm based on four basic multiplication algorithms, we give a fast software implementation of length-adaptive privacy amplification. "Length-adaptive" indicates that the implementation of privacy amplification automatically adapts to different lengths of input blocks. When the lengths of the input blocks are 1 Mbit and 10 Mbit, the speed of privacy amplification can be as fast as 14.86 Mbps and 10.88 Mbps, respectively. Thus, it is practical for GHz or even higher repetition frequency QKD systems.

Keywords: length-adaptive privacy amplification, multiplication algorithms, quantum key distribution

PACS: 03.67.Hk, 03.67.Dd, 03.67.–a **DOI:** 10.1088/1674-1056/23/9/090310

1. Introduction

The goal of quantum key distribution (QKD) is to distribute random secret keys between two distant legitimate parties, customarily called Alice and Bob, in the presence of an eavesdropper Eve. The best known protocol[1] for QKD was proposed by Bennett and Brassard in 1984. Generally, QKD contains four phases: quantum communication, public discussion, key reconciliation, and privacy amplification. In the first phase, Alice sends Bob a series of quantum states through a quantum channel, then Bob measures the quantum states that he receives. The key in this phase is termed the raw key. In the public discussion phase, Alice and Bob announce their chosen bases through a public classical authenticated channel, and only the qubits that have the same bases are retained. After this phase, Alice and Bob obtain the correlated classical bits of the same length called the sifted key. In the key reconciliation phase, Alice and Bob have to reconcile the errors in the sifted key due to the unavoidable tampering in the quantum channel. The key in this phase is called the reconciled key. Finally, to eliminate the partial information gained by Eve, Alice and Bob use universal hash functions to compress the reconciled key. The key after this phase is called the secret key.

Since 1984, many theoretical and experimental achievements in QKD have been made.[2–6] Several research groups have realized high-speed QKD systems.[2,3,7] For high-speed QKD systems, privacy amplification (PA) is a bottleneck due to the fact that the input length should be larger than at least a hundred kilobits[8] to reduce the finite size effect.[8–12] To fulfill the high-speed requirement and to reduce the finite size effect, we propose a universal method to speed up the PA process at large input lengths on different QKD systems.

PA[13,14] allows Alice and Bob to distill shorter but highly secret keys from a common random string that is only partially secret. A publicly known universal class of hash functions[15,16] is used to perform the PA process. To reduce the finite size effect in distilling secure keys,[8,12] the lengths of input blocks for PA should be larger than a hundred kilobits, and this leads to large lengths of processing blocks. There are several universal classes of hash functions[17] suitable for PA. Many QKD groups[2,18,19] utilize the Toeplitz matrix[20] for this task. It can either be realized based on hardware (for its simplicity and parallel feature), or software which can be accelerated by means of fast Fourier transform (FFT) or number theoretic transform.[17,21]

Instead of choosing the Toeplitz matrix, we choose a simple multiplicative universal class of hash functions[17] to perform PA. By constructing an optimal multiplication algorithm based on four basic multiplication algorithms, we give a fast software implementation of length-adaptive PA with good portability and universality, which can be easily trans-

*Project supported by the National Basic Research Program of China (Grant Nos. 2011CBA00200 and 2011CB921200) and the National Natural Science Foundation of China (Grant Nos. 60921091 and 61101137).

†Corresponding author. E-mail: yinzheqi@mail.ustc.edu.cn; kooky@mail.ustc.edu.cn

© 2014 Chinese Physical Society and IOP Publishing Ltd

planted to different repetition frequency QKD systems. The PA implementation is performed by a single thread computer program, which is written in C++. The experimental platform is a personal computer with Intel (R) Core (TM) i3-3220 @3.30 GHz and 4.00 GB RAM running 64-bit Windows 8. "Length-adaptive" indicates that the PA implementation automatically adapts to various lengths of input blocks, which enables users to adjust the input length according to their specific requirements. In principle, there is no upper bound on the input length, and the time overhead of PA is not affected by the shrink factor (the ratio of the length of the secret keys to the length of the reconciled keys). The proposed scheme features shorter time overhead than that in Ref. [22] which chooses the Toeplitz matrix to perform PA. When the processing block lengths are 1 Mbit and 10 Mbit, the speeds of the length-adaptive PA scheme can be up to 14.86 Mbps and 10.88 Mbps, respectively. In this sense, the proposed PA scheme is adequate for practical GHz or even higher repetition frequency QKD systems.

The rest of this paper is organized as follows. In Section 2, privacy amplification and related multiplication algorithms are presented. Section 3 gives the scheme of the optimal multiplication algorithm. Section 4 discusses the construction of the optimal multiplication algorithm. A performance analysis of the length-adaptive privacy amplification scheme is presented in Section 5. Finally, Section 6 concludes the paper.

2. Privacy amplification and related algorithms

In order to eliminate the partial information that Eve may know about the keys, Alice and Bob have to randomly choose a hash function from a publicly known universal class to compress the keys shared between them. In practice, Alice sends some information that can describe the hash function to Bob, and they use the same hash function to perform the PA process.

There are a few universal classes of hash functions to perform the PA process.[17] One of them has the following form

$$H_{k,l} = \{h_{a,b} | a,b \in Z_{2^k}, \quad \gcd(a,2) = 1, b \neq 0\}, \tag{1}$$

where k and l ($k < l$) are two positive integers, and $h_{a,b}$ is given by

$$h_{a,b}(x) = \left\lfloor (ax+b \bmod 2^k)/2^{k-l} \right\rfloor, \quad x \in Z_{2^k}. \tag{2}$$

The hash function $h_{a,b}$ indicates the l most significant bits of the function $ax+b$ which is calculated in the ring of integers modulo 2^k. For binary strings, $ax+b \bmod 2^k$ denotes the rightmost k bits of $ax+b$, and $\lfloor (ax+b \bmod 2^k)/2^{k-l} \rfloor$ denotes the leftmost l bits of $(ax+b \bmod 2^k)$. Hence the modulo and division operations are not actually needed to do. Compared with the addition operation, the multiplication operation is more complicated and nearly consumes all the time overhead of the entire process. Fortunately, there are some multiplication algorithms[21,23–25] to boost the multiplication operation according to the lengths of the two integers that are to be multiplied. Reference [25] is suitable for astronomically large numbers and is not considered in this paper.

A brief description of some useful multiplication algorithms related to the proposed PA scheme is given below.

2.1. Grade-school algorithm

Given two n-digit integers a and b in some base B ($B \in Z$)

$$a = \sum_{i=0}^{n-1} a_i B^i, \tag{3}$$

$$b = \sum_{j=0}^{n-1} b_j B^j, \tag{4}$$

the product of a and b can be written as

$$c = \sum_{k=0}^{2n-1} c_k B^k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j B^{i+j}, \tag{5}$$

where a_i, b_j, $c_i \in Z_B$. The time complexity of grade-school (G) algorithm is $O(n^2)$.

2.2. Karatsuba algorithm

Considering two large n-digit integers a and b in some base B ($B \in Z$), a and b are divided into two parts of equal length or the high part shorter than the low part. Let $m = \lceil n/2 \rceil$, a and b can be written as

$$a = a_1 B^m + a_0, \tag{6}$$

$$b = b_1 B^m + b_0, \tag{7}$$

where $a_0, b_0 \in Z_{B^m}$ and $a_1, b_1 \in Z_{B^{(n-m)}}$. According to Karatsuba (K) algorithm,[23] the product of a and b can be denoted as

$$\begin{aligned} c = {} & a_1 b_1 B^{2m} + [(a_1 + a_0)(b_1 + b_0) \\ & - a_1 b_1 - a_0 b_0] B^m + a_0 b_0. \end{aligned} \tag{8}$$

The product of a and b can be computed in only three multiplications at the cost of some extra additions and subtractions. The time complexity of K algorithm is $O(n^{\log_2 3})$.

2.3. Toom–Cook 3-way algorithm

Toom–Cook 3-way (TC) algorithm,[24] in some sense, is a generalization of K algorithm. If the length of two large numbers a and b in some base B ($B \in Z$) is n, each number will be divided into three parts of equal length or the most significant part shorter than the other two. Anyway, the lengths of the three parts should be as close as possible, a and b can be written as

$$a = a_2 B^{2m} + a_1 B^m + a_0, \tag{9}$$

$$b = b_2 B^{2m} + b_1 B^m + b_0. \tag{10}$$

In view of two polynomials $A(x)=a_2x^2+a_1x+a_0$ and $B(x)=b_2x^2+b_1x+b_0$, a and b are the value of $A(x)$ and $B(x)$ at the point $x=B^m$, where $m=\lfloor (n+2)/3 \rfloor$. Let $C(x)$ be the product of $A(x)$ and $B(x)$. Then, $C(x)$ can be denoted as

$$C(x)=c_4x^4+c_3x^3+c_2x^2+c_1x+c_0. \tag{11}$$

The product of a and b is the value of $C(x)$ at the point $x=B^m$. Once the five coefficients of $C(x)$ are determined, the product of a and b can be determined. To ascertain the five coefficients, an evaluation of $A(x)$ and $B(x)$ at five different points is made, then point-wise multiplications of the five pairs and a following interpolation are performed.

There are only five multiplications in this algorithm (in fact, there are some extra operations in the interpolation step like some additions, subtractions, and divisions which can be disposed quickly). TC algorithm gives a time complexity of $O(n^{\log_3 5})$.

2.4. Schönhage–Strassen algorithm

Schönhage–Strassen (SS) algorithm[21,26] plays a prominent part in multiplying two very large integers. The core of this algorithm lies in FFT in some algebraic rings which can speed up multiplications and ensure accurate multiplication results.

Assume $n=2^kM$ $(k,M\in Z)$. If this equation is not satisfied, the integers to be multiplied can be padded with certain zeros. The product of two very large n-digit integers can be reduced to polynomial multiplication in $Z_{n'}[x]$ mod $(x^{2^k}+1)$ for some $n'\geq 2M+k$ which can be done by two forward FFTs, point-wise multiplications, and a backward FFT.

Similar to TC algorithm, SS algorithm contains five steps: split, evaluation (forward FFT), point-wise multiplication, interpolation (backward FFT), and combination. Due to the use of FFT, SS algorithm can be efficiently performed in the $O(n\log n\log\log n)$ time complexity.

3. Scheme of the optimal multiplication algorithm

K algorithm, TC algorithm, and SS algorithm are divide and conquer (D&C) strategies. Generally, the D&C strategy solves a problem by three steps. (i) Divide the problem to be solved into two or more small subproblems of the same or related type. (ii) Conquer subproblems recursively according to step (i), until the subproblems are so tiny that they can be or must be solved directly. (iii) Combine the results of subproblems into the result of the original problem.

Intuitively, SS algorithm should be recursively adopted to solve a multiplication problem due to its best asymptotic time behavior among the four algorithms in Section 2. However, it is a mistake to apply only one multiplication algorithm recursively until tiny subproblems.[27] Take SS algorithm and G algorithm for example. SS algorithm is the most complicated but best asymptotic time algorithm, while G algorithm is the simplest but worst asymptotic time algorithm. To compute small or tiny multiplication subproblems, G is faster due to its simplicity, while SS algorithm is slower due to its complexity. Hence, to solve the subproblems at different recursive levels efficiently, a tradeoff between the asymptotic time behavior and the simplicity has to be considered.

A multiplication algorithm is considered to be optimal if there is no algorithm that can perform faster. For an optimal multiplication algorithm to compute the product of two large numbers, it is very necessary to adopt appropriate algorithms at different recursive levels, so that the original problem can be computed efficiently. In other words, the optimal algorithm is a combination of several multiplication algorithms, which can adopt a different algorithm for the next level of its recursion, a third algorithm for the third level, and so on.

4. Construction of the optimal multiplication algorithm

For the optimal multiplication algorithm at different recursive levels, the crucial point lies in finding appropriate switching points where one algorithm is switched to another. Theoretically, the number of mathematical operations is counted in every algorithm, then the appropriate switching points can be determined accordingly. In practice, the number of a specific algorithm's mathematical operations is closely related to its running time overhead. Hence, a simple but effective method to determine the switching point is to find the point where the time overhead is shortest. Once the appropriate switching points are determined, an optimal multiplication algorithm can be constructed accordingly. The searches for the switching points that optimize the multiplication algorithm are described as follows.

4.1. Switching point between K algorithm and G algorithm

The algorithm combining K algorithm and G algorithm is denoted as K_G algorithm. Without loss of generality, assume $n=2^k$ for some integer k. For K algorithm, the recurrence relation can be written as $T_{\mathrm{K}}(n)=3T_{\mathrm{K}}(n/2)+\Theta(n)$. Considering K_G algorithm, the recurrence relation[2] can be expressed as

$$T_{\mathrm{K_G}}(n)=\begin{cases} T_{\mathrm{G}}(n), & n\leq P_{\mathrm{K_G}}, \\ 3T_{\mathrm{K_G}}(n/2)+\Theta(n), & n>P_{\mathrm{K_G}}, \end{cases} \tag{12}$$

where $T_{\mathrm{K}}(n)$, $T_{\mathrm{G}}(n)$, and $T_{\mathrm{K_G}}(n)$ denote the running time of two n-bit integers' multiplication using K algorithm, G algorithm, and K_G algorithm respectively, $\Theta(n)$ denotes the linear running time of division and combination, and $P_{\mathrm{K_G}}$ is the appropriate switching point between K algorithm and G algorithm.

K_G algorithm has an asymptotic complexity of $O(n^{\log_2 3})$, which is the same as K algorithm. However, for limited lengths of integers, K_G performs faster than K algorithm due to the fact that the constant coefficients ignored in O-notation of K_G algorithm is smaller than the coefficients ignored in K algorithm.

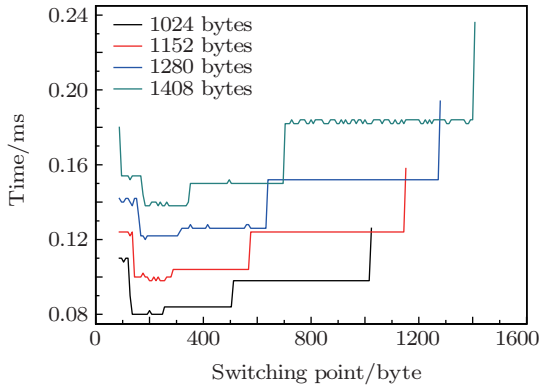


Fig. 1. (color online) The experimental result of finding $P_{\text{K_G}}$. The time overhead with respect to different tested switching points is recorded when the input lengths are 1024, 1152, 1280, and 1408 bytes, respectively. The last point for every curve denotes the time overhead of the multiplication process using only G algorithm, and other points denote the time overhead using K_G algorithm. The switching points between 184 and 240 bytes exhibit short time overhead in all four situations. Hence, $P_{\text{K_G}}$ between K algorithm and G algorithm should be chosen between 184 and 240 bytes.

To find $P_{\text{K_G}}$, every integer s ($1 \le s \le n$) is tested as the switching point, and the time overhead for every s is correspondingly recorded. It can be seen from the recurrence relation that the multiplication process uses G algorithm when $s = n$. This procedure is repeated for several different n. The tested switching point, which features short time overhead for different n, should be $P_{\text{K_G}}$. The experimental result of finding $P_{\text{K_G}}$ is plotted in Fig. 1.

As shown in Fig. 1, the switching points from 184 to 240 bytes exhibit low time overhead for all four samples. Hence, $P_{\text{K_G}}$ where K algorithm is switched to G algorithm can be randomly chosen between 184 and 240 bytes. In this paper, $P_{\text{K_G}}$ is chosen as 224 bytes.

4.2. Switching point between TC algorithm and K_G algorithm

The algorithm combining TC algorithm and K_G algorithm is denoted as TC_K_G algorithm. For simplicity, let $n = 3^k$ for some integer k. For TC algorithm, the recurrence relation can be expressed as $T_{\text{TC}}(n) = 5T_{\text{TC}}(n/3) + \Theta(n)$. Considering TC_K_G algorithm, the recurrence relation[28] can be written as

$$T_{\text{TC_K_G}}(n) = \begin{cases} T_{\text{K_G}}(n), & n \le P_{\text{TC_K_G}}, \\ 5T_{\text{TC_K_G}}(n/3) + \Theta(n), & n > P_{\text{TC_K_G}}, \end{cases} \quad (13)$$

where $T_{\text{TC}}(n)$ and $T_{\text{TC_K_G}}(n)$ denote the running time of two n-bit integers' multiplication using TC algorithm and TC_K_G algorithm respectively, $\Theta(n)$ denotes the linear running time of division and combination operations, and $P_{\text{TC_K_G}}$ is the appropriate switching point between TC algorithm and K_G algorithm.

TC_K_G algorithm has an asymptotic complexity of $O(n^{\log_3 5})$, which is the same as TC algorithm. Similarly, TC_K_G algorithm performs faster than TC algorithm practically, which can be explained by the small constant coefficients ignored in O-notation of TC_K_G algorithm.

The method of finding $P_{\text{TC_K_G}}$ is very similar to $P_{\text{K_G}}$. To find $P_{\text{TC_K_G}}$, every integer s ($1 \le s \le n$) is tested as the switching point between TC algorithm and K_G algorithm, and the time overhead for every s is correspondingly recorded. It can be seen from the recurrence relation that the multiplication process uses K_G algorithm when $s = n$. This procedure is repeated for several different n. The tested switching point, which features short time overhead for different n, should be $P_{\text{TC_K_G}}$. The experimental result of finding $P_{\text{TC_K_G}}$ is plotted in Fig. 2. As shown in Fig. 2, it can be seen that the switching points from 440 to 904 bytes exhibit short time overhead for all four test samples. Hence, the appropriate switching point $P_{\text{TC_K_G}}$ where TC algorithm is switched to K_G algorithm should be chosen from 440 to 904 bytes. In this paper, $P_{\text{TC_K_G}}$ is randomly chosen as 824 bytes.

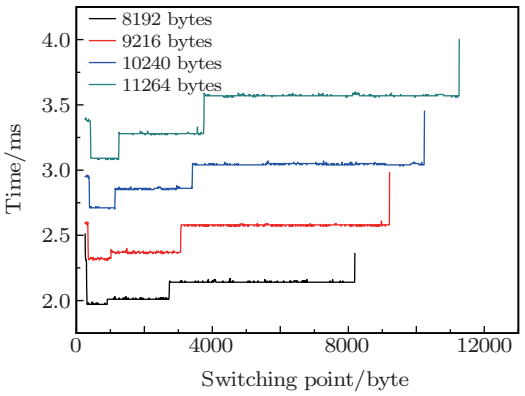


Fig. 2. (color online) The experimental result of finding $P_{\text{TC_K_G}}$. The time overhead with respect to different switching points is recorded when the input lengths are 8192, 9216, 10240, and 11264 bytes, respectively. The last point for every curve denotes the time overhead of the multiplication process using only K_G algorithm, and other points denote the time overhead using TC_K_G algorithm. The switching points from 440 to 904 bytes show short time overhead for every curve. Hence $P_{\text{TC_K_G}}$ between TC algorithm and K_G algorithm should be chosen from 440 to 904 bytes.

4.3. Switching point between SS algorithm and TC_K_G algorithm

The algorithm combining SS algorithm and TC_K_G algorithm is denoted as SS_TC_K_G algorithm. Assume the integer length $n = 2^k M$ for some integers k and M. If n is not satisfied with this equation, the integers to be multiplied can be padded with certain zeros. For SS algorithm, the recurrence relation can be expressed as $T_{\text{SS}}(n) = 2^k T_{\text{SS}}(n/2^k) + \Theta(n)$. Considering SS_TC_K_G algorithm, the recurrence relation[28] is

$$T_{\mathrm{SS_TC_K_G}}(n)=\begin{cases}T_{\mathrm{TC_K_G}}(n), & \text{if } n\le P_{\mathrm{SS_TC_K_G}},\\ 2^k T_{\mathrm{SS_TC_K_G}}(n/2^k)+\Theta(n), & \text{if } n>P_{\mathrm{SS_TC_K_G}},\end{cases} \tag{14}$$

where $T_{\mathrm{SS}}(n)$ and $T_{\mathrm{SS_TC_K_G}}(n)$ denote the running time of two n-bit integers' multiplication using SS algorithm and SS_TC_K_G algorithm respectively, $\Theta(n)$ denotes the running time of division and combination operations, and $P_{\mathrm{SS_TC_K_G}}$ is the appropriate switching point between SS algorithm and TC_K_G algorithm.

SS_TC_K_G algorithm has an asymptotic time complexity of $O(n\log n\log\log n)$, which is the same as SS algorithm. In practice, SS_TC_K_G algorithm performs faster than SS algorithm, which can be explained by the small constant coefficients ignored in O-notation of SS_TC_K_G algorithm. SS_TC_K_G algorithm is the optimal multiplication algorithm to be used in the PA process.

The method of finding $P_{\mathrm{SS_TC_K_G}}$ is similar to $P_{\mathrm{K_G}}$ and $P_{\mathrm{TC_K_G}}$, but the analysis is slightly different from them. To find $P_{\mathrm{SS_TC_K_G}}$, every integer s ($1\le s\le n$) is tested as the switching point between SS algorithm and TC_K_G algorithm, and the time overhead for every s is correspondingly recorded. It can be seen from the recurrence relation that the multiplication process uses TC_K_G algorithm when $s=n$, and this is very helpful to find $P_{\mathrm{SS_TC_K_G}}$. The experimental result of finding $P_{\mathrm{SS_TC_K_G}}$ is plotted in Fig. 3.

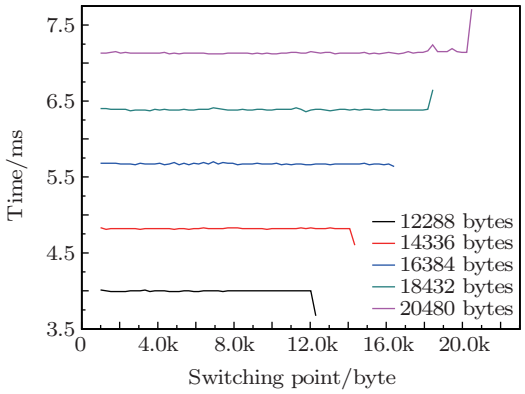


Fig. 3. (color online) The experimental result of finding $P_{\mathrm{SS_TC_K_G}}$. The time overhead with respect to different switching points is recorded when the input lengths are 12288, 14336, 16384, 18432, and 20480 bytes, respectively. The last point for every curve denotes the time overhead of the multiplication process using only TC_K_G algorithm, and other points denote the time overhead using SS_TC_K_G algorithm. When the integer lengths are 12288 and 14336 bytes, compared with TC_K_G algorithm, using SS algorithm at the first recursive level is not worthy. When the integer length is 16384 bytes, TC_K_G algorithm and SS_TC_K_G algorithm exhibit almost the same time overhead. When the integer length is larger than 16384 bytes, SS_TC_K_G algorithm begins to show its advantage. Hence, $P_{\mathrm{SS_TC_K_G}}$ should be 16384 bytes.

As shown in Fig. 3, the last point for every curve denotes the time overhead of the multiplication process using TC_K_G algorithm. When the input length is less than 16384 bytes, compared with TC_K_G algorithm, using SS algorithm at the first recursive level is more time consuming. When the input length is larger than 16384 bytes, SS_TC_K_G algorithm begins to show its advantage. When the input length is 16384 bytes, TC_K_G algorithm and SS_TC_K_G algorithm exhibit almost the same time overhead. Hence, the appropriate switching point $P_{\mathrm{SS_TC_K_G}}$ between SS algorithm and TC_K_G algorithm should be 16384 bytes.

To illustrate the superiority of the optimal combined multiplication algorithm, we make a comparison of G algorithm, K algorithm, TC algorithm, SS algorithm, and SS_TC_K_G algorithm. The experimental result is given in Fig. 4. As shown in Fig. 4, SS_TC_K_G algorithm shows the best time feature, and can adapt to different input lengths. The stepwise behavior of SS_TC_K_G algorithm can be explained by the FFT efficiency[26] in SS algorithm. Obviously the length-adaptive SS_TC_K_G algorithm is the optimal multiplication algorithm that can be used to perform fast implementation of length-adaptive PA.

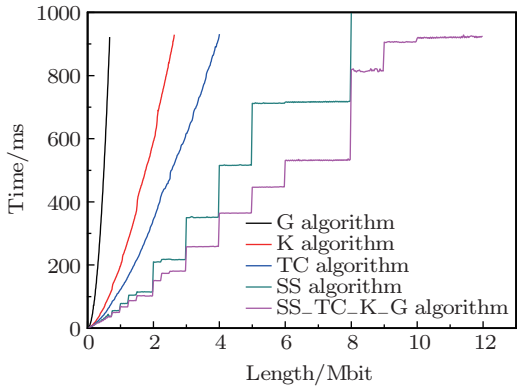


Fig. 4. (color online) SS_TC_K_G algorithm shows the best time feature, and can adapt to different input lengths. Obviously the length-adaptive SS_TC_K_G algorithm is the optimal multiplication algorithm that can be used to perform fast implementation of length-adaptive PA.

5. Performance analysis of the length-adaptive privacy amplification scheme

To perform PA, the hash function Eq. (2) requires the multiplication operation and the addition operation. We adopt SS_TC_K_G algorithm to perform the multiplication in PA. As long as the input lengths are larger than at least a hundred kilobits,[8] it can reduce the finite size effect, and the switches of multiplication algorithms are intermediate results which do not involve the reconciled keys. The time overhead of the entire PA process at different input lengths is tested. The experimental result of the time overhead with respect to different input lengths for the length-adaptive PA scheme is given in Fig. 5.

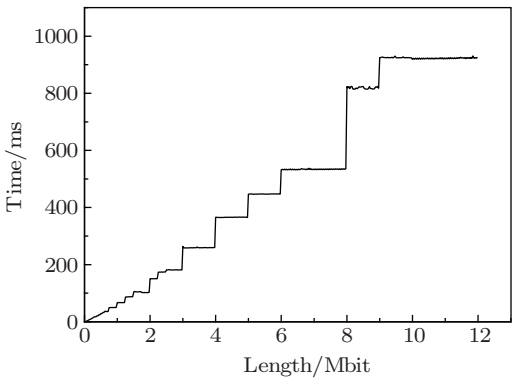


Fig. 5. The experimental result of the time overhead with respect to different input lengths for the length-adaptive PA scheme.

The curve in Fig. 5 is very similar to the SS_TC_K_G algorithm curve in Fig. 4. Nevertheless, the two curves are different. Figure 5 denotes the time overhead of the entire PA process which contains the multiplication operation and the addition operation, while the SS_TC_K_G algorithm curve in Fig. 4 denotes the time overhead of the multiplication operation. The similarity between these two curves illustrates that the multiplication operation consumes nearly all the time overhead of the entire PA process, so it is very necessary to construct an optimal multiplication algorithm to speed up the multiplication operation, and thus the entire PA process.

Even implemented based on software, the proposed scheme is completely comparable to hardware schemes. For example, the average processing speed at 750 kbit is 7.71 Mbps for NEC–NICT system whose privacy amplification module is based on FPGA,[22] while the proposed PA scheme achieves the speed of 20.3 Mbps. Furthermore, when the input lengths are 1 Mbit and 10 Mbit, the proposed length-adaptive PA scheme can achieve the speed of 14.86 Mbps and 10.88 Mbps, respectively.

Finally, the comparison of the proposed PA scheme and reference [29] is shown in Table 1. It can be seen that the proposed PA scheme exhibits an obvious advantage at large input lengths. Due to the large input lengths and high processing speed, this scheme is adequate for practical GHz or even higher repetition frequency QKD systems.

Table 1. Comparison of PA schemes.

Input length bit	512 k	1 M	2 M	4 M	8 M	10 M	12 M
Ref. [29] Mbps	5.95	4.92	4.61	3.70	3.61	–	–
Proposed scheme Mbps	17.42	14.86	13.26	10.94	9.71	10.88	12.98

6. Conclusion

In conclusion, by introducing four basic multiplication algorithms and constructing an optimal multiplication algorithm, we give a fast implementation of length-adaptive privacy amplification based on software in quantum key distribution. The proposed length-adaptive PA scheme has three main advantages. (i) High speed. The speed of the proposed PA scheme is high enough to be adopted in practical GHz or even higher repetition frequency QKD systems. (ii) Good portability. Due to the length-adaptive trait and the software implementation, it can be easily transplanted to different QKD systems. (iii) Low cost. Compared with PA based on hardware, the cost of the proposed PA scheme based on software is relatively low. This work is promising in present high-speed QKD systems.

References

[1] Bennet C H and Brassard G 1984 *Processings of IEEE International Conference on Computers, Systems and Singnal Processing* p. 175
[2] Stucki D, Walenta N, Vannel F, Thew R T, Gisin N, Zbinden H, Gray S, Towery C R and Ten S 2009 *New J. Phys.* **11** 075003
[3] Yuan Z L, Dixon A R, Dynes J F, Sharpe A W and Shields A J 2009 *New J. Phys.* **11** 045019
[4] Li H W, Yin Z Q, Wang S, Bao W S, Guo G C and Han Z F 2011 *Chin. Phys. B* **20** 100306
[5] Xu F X, Wang S, Han Z F and Guo G C 2010 *Chin. Phys. B* **19** 100312
[6] Zhang Y, Wang S, Yin Z Q, Chen W, Liang W Y, Li H W and Guo G C 2012 *Chin. Phys. B* **21** 100307
[7] Wang S, Chen W, Guo J F, Yin Z Q, Li H W, Guo G C and Han Z F 2012 *Opt. Lett.* **37** 1008
[8] Hasegawa J, Hayashi M, Hiroshima T, Tanaka A and Tomita A 2007 *arXiv*:0705.3801 [astro-ph]
[9] Scarani V and Renner R 2008 *Phys. Rev. Lett.* **100** 200501
[10] Tomamichel M, Lim C C W, Gisin N and Renner R 2012 *Nat. Commun.* **3** 634
[11] Lucamarini M, Patel K A, Dynes J F, Fröhlich B, Sharpe A W, Dixon A R, Yuan Z L, Penty R V and Shields A J 2013 *Opt. Express* **21** 24550
[12] Li H W, Zhao Y B, Yin Z Q, Wang S and Han Z F 2009 *Opt. Commun.* **282** 4162
[13] Bennet C H, Brassard G and Robert J M 1988 *Siam J. Comput.* **17** 210
[14] Bennet C H, Brassard G, Crepeau C and Maurer U M 1995 *IEEE Trans. Inform. Theory* **41** 1915
[15] Carter J L and Wegman M N 1979 *J. Comput. Syst. Sci.* **18** 143
[16] Wegman M N and Carter J L 1981 *J. Comput. Syst. Sci.* **22** 265
[17] Assche G V 2006 *Quantum Cryptography and Secret-key Distillation* (1st edn.) (Cambridge: Cambridge University Press) pp. 104–107
[18] Eraerds P, Walenta N, Lergre M, Gisin N and Zbinden H 2010 *New J. Phys.* **12** 063027
[19] Tanaka A, Fujiwara M, Yoshino K, Takahashi S, Nambu Y, Tomita A, Miki S, Yamashita T, Wang Z, Sasaki M and Tajima A 2012 *IEEE J. Quantum Electron.* **48** 542
[20] Krawczyk H 1994 *Proc. Adv. Cryptol.-CRYPTO* **839** 129
[21] Schönhage A and Strassen V 1971 *Computing* **7** 281
[22] Sasaki M, Fujiwara M, Klaus W, *et al.* 2011 *Opt. Express* **19** 10387
[23] Karatsuba A and Ofman Y 1962 *Proceedings of the USSR Academy of Sciences*, Vol. 145 p. 293
[24] Toom A L 1963 *Soviet Mathematics* **3** 714
[25] Furer M 2009 *Siam J. Comput.* **39** 979
[26] Gaudry P, Kruppa A and Zimmermann P 2007 *Proceedings of ISSAC* 2007 p. 167
[27] Bernstein D J 2001 *Multidigit Multiplication for Mathematicians* http://cr.yp.to/papers.html#m3
[28] Cormen T H, Leiserson C E, Rivest R L and Stein C 2009 *Introduction to Algorithms* (3rd edn.) (London: MIT Press) pp. 65–74
[29] Pacher C, Lechner G, Portmann C, Maurhart O and Peev M 2012 *QCRYPT*, September, 10–14, 2012 Singapore