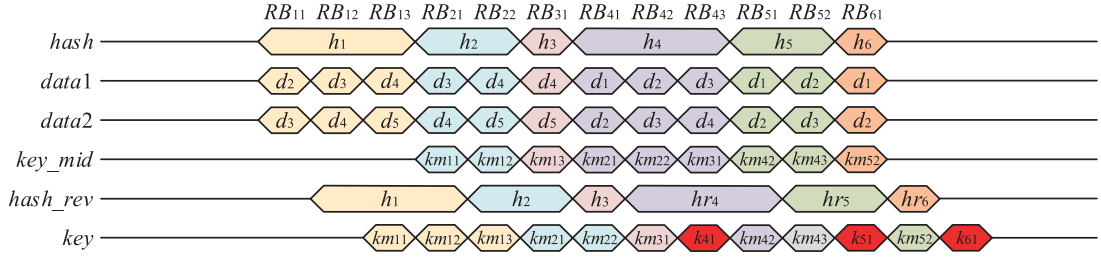
IEEE Photonics Journal
An IEEE Photonics Society Publication

Open Access

FPGA-Based Implementation of Size-Adaptive Privacy Amplification in Quantum Key Distribution

Volume 9, Number 6, December 2017

Shen-Shen Yang
Zeng-Liang Bai
Xu-Yang Wang
Yong-Min Li



DOI: 10.1109/JPHOT.2017.2761807
1943-0655 © 2017 IEEE

IEEE photonics SOCIETY

IEEE

FPGA-Based Implementation of Size-Adaptive Privacy Amplification in Quantum Key Distribution

Shen-Shen Yang, Zeng-Liang Bai, Xu-Yang Wang, and Yong-Min Li

State Key Laboratory of Quantum Optics and Quantum Optics Devices, Collaborative Innovation Center of Extreme Optics, Institute of Opto-Electronics, Shanxi University, Taiyuan 030006 China

DOI:10.1109/JPHOT.2017.2761807

Manuscript received September 8, 2017; accepted October 7, 2017. Date of publication October 10, 2017; date of current version October 24, 2017. This work was supported in part by the Key Project of the Ministry of Science and Technology of China under Grant 2016YFA0301403, in part by the National Natural Science Foundation of China under Grants 61378010 and 11504219, in part by the Shanxi 1331KSC, and in part by the Program for the Outstanding Innovative Teams of Higher Learning Institutions of Shanxi. Corresponding author: Yong-Min Li (e-mail: yongmin@sxu.edu.cn).

Abstract: In a quantum key distribution (QKD) system, privacy amplification (PA) is an essential procedure that can effectively eliminate the leaked information to an eavesdropper and distill a secret key. The processing speed of the PA algorithm inevitably affects the final key rate of the QKD system. We propose a high-speed PA algorithm based on field-programmable gate array (FPGA), where the matrix multiplications are divided into a number of rhomboid-block operations. The implementation of the PA reduces significantly the required number of FPGA Block RAMs. Meanwhile, it automatically adapts to different lengths of input and output blocks. Finally, we verified the correctness and high-speed of the algorithm when implemented in a Xilinx Virtex-7 FPGA.

Index Terms: Quantum key distribution, privacy amplification, hardware implementation, field programmable gate array, size-adaptive.

1. Introduction

Quantum key distribution (QKD) is the first quantum information task to reach the level of mature technology, and enables the sharing of an unconditionally secure key string between two legitimate parties (Alice and Bob). Over the past three decades, various protocols have been proposed, and great progress has been made in terms of experiments [1]–[9]. A QKD system is typically divided into four steps: (1) Quantum state preparation, distribution, and measurement; (2) Data sifting and parameter estimation; (3) Information reconciliation in which Alice and Bob correct errors in sifted keys to obtain identical corrected keys; and (4) Privacy amplification (PA) [10], [11], which eliminate the leaked information by using universal classes of hash functions to map long-corrected keys to shorter final keys.

For a high-speed QKD system, the processing speed of PA in software has been unable to match the raw key generation speed. Although some algorithms, such as the number theoretic transform (NTT) [12] and optimal multiplication algorithm [13], enable us to perform fast matrix multiplications, it is desirable to find an even faster approach. One way to achieve this is by using hardware-based acceleration. Field-programmable gate arrays (FPGA) features powerful parallel

operation and straightforward matrix multiplications. Owing to their faster operation rate and better time efficiency compared with conventional algorithms executed through software, FPGA are very attractive when designing prototypes and manufacturing small-production-run devices and their in-system programmability makes them substantially cost-effective [14]. Although we can also use GPU or Coprocessor to accelerate the data processing [15], [16], FPGA is good at binary operations and can configure the degree of parallelism more flexibly. So that it facilitates control over the trade-off between data throughput and hardware resource requirements [17]. In addition, FPGA energy consumption is much lower than that of GPU and Coprocessor. However, one disadvantage of FPGA is that hardware resources are limited when applied in a complete QKD system. Therefore, it is a major engineering challenge to optimize and reduce the number of required hardware resources.

In terms of FPGA-based privacy amplification, several works have been reported [18]–[21]. The authors in [19] proposed an approach that implemented PA algorithm with "block operations" in FPGA. In this method, the hash functions were divided into blocks that operated on a small matrix of 40×40 bits. In terms of speed, four clock cycles were required to complete the reading and multiplication operation of one matrix block. In addition, that architecture required a large amount of storage on-chip. The operating speed can be improved further by using a pipeline design. For example, the authors in [20], [21] showed such a design to fully exploiting every clock cycle, and the data could be stored in an off-chip memory (DDR2-RAM) instead of the on-chip Block RAM (BRAM).

In a QKD system, large block of hash functions are needed to ensure a high secure key rate due to finite size effects in the security analysis [22], [23], and the storage of the these hash functions consumes large amounts of storage space. However, if the hash functions are stored in the off-chip memory, data read and write will increase the complexity of QKD system control. In this paper, we propose a novel high-speed PA algorithm that significantly reduces the storage requirements so that on-chip BRAM can be used. Another feature of our scheme is size-adaptive, which means that the PA can adapt to various lengths of input and output blocks automatically.

The rest of this paper is organized as follows. In Section 2, we present the principle and advantages of the proposed PA algorithm. In Section 3, we describe an FPGA-based implementation of the proposed PA algorithm. In Section 4, we present the relationship between the processing speed and the requirements of the hardware resources. We also show a comparison between several FPGA-based hardware implementations, including our own. In Section 5, we provide a brief conclusion.

2. Underlying Principles of the Algorithm

2.1. Hash Functions

In the case of the PA algorithm, universal classes of hash functions [24] with the following requirements should be utilized: (1) the closer it is to universality, the better the secrecy of the resulting key; (2) classes with large input and output sizes; and (3) evaluation of the hash function should not consume much time.

Toeplitz hashing is a particular class of universal hash functions [25] that meet the requirements for the PA [18], [26]. Toeplitz hashing uses a diagonal-constant matrix, which can be constructed using its first row and first column. This specific structure reduces the number of random bits required when constructing a random matrix. For example, the required number of random bits will be reduced from MN to $M + N - 1$ when an $M \times N$ diagonal-constant matrix is constructed.

2.2. Rhomboid-block Operation

The secret key rate ΔI of a QKD system can be written as:

$$\Delta I = \beta I_{AB} - \chi_{BE}, \tag{1}$$

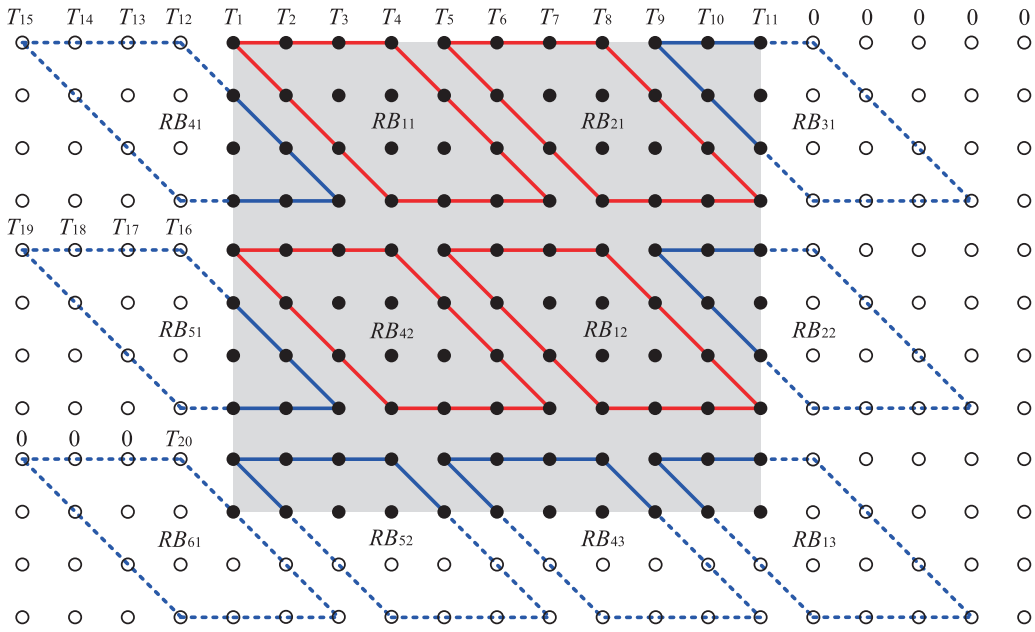


Fig. 1. Principle of the Algorithm. The black dots represents the elements of Toeplitz matrix; the circles represents the padding elements. $L_k = 10$, $L = 11$, $p = 4$, $M = 12$, $N = 20$, $m = 3$, $n = 4$, $s = 6$. See the main text for the details of the parameters.

where β denotes the reconciliation efficiency, I_{AB} is the classical mutual information between two legitimate parties, and χ_{BE} is the leakage information. The length of the final key is

$$L_k = \Delta I \cdot L, \tag{2}$$

where L denotes the length of the corrected keys. The compression ratio is defined by:

$$r = L_k / L. \tag{3}$$

From (2), the Toeplitz hashing is an $L_k \times L$ matrix. Since Toeplitz hashing has a regular diagonal structure, we propose the concept of a "Rhomboid-block (RB)" operation to perform the matrix multiplication. The RB operation consists of p parallel multiply-accumulator (PMAC) units, in which each unit performs a p single-bit multiplication over a Galois field $GF(2)$. More specifically, it combines the bits of Toeplitz hashing with the bits of the corrected keys. The single-bit multiplication of every bit in a PMAC unit is performed at the same time.

We now describe the PA algorithm in detail. To ensure that the RB operation can involve all elements of a Toeplitz hashing, we must reconstitute the Toeplitz hashing by inserting some padding elements. An example is shown in Fig. 1. A 10×11 Toeplitz hashing is reconstituted into a new 12×20 matrix. An $L_k \times L$ Toeplitz hashing is reconstituted into a new $M \times N$ matrix, where $M = \lceil L_k/p \rceil p$, $N = \lceil L/p \rceil p + 2p$, following the diagonal rule that each descending diagonal from upper left to lower right is constant. According to this rule, $M - L_k$ rows are padded after the last row, p columns are padded in front of the first column, and $N - L - p$ columns are padded after the last column, respectively. Finally, the rest elements that cannot be covered by the diagonal rule are padded with zeros. The new matrix is further divided into an $m \times n$ RB matrixe, where $m = \lceil L_k/p \rceil$, $n = \lceil L/p \rceil + 1$. As shown in Fig. 1 and (4), a 10×11 Toeplitz hashing is finally reconstituted into a 3×4 RB matrix following the above methods. The reconstitution of the Toeplitz hashing simplifies significantly the calculation when implemented with FPGA. The corrected keys also need to be reconstituted following the rule that p zeros are added in the front of the first bit and $N - L - p$

zeros are padded after the last bit.

$$\begin{bmatrix} T_1 & T_2 & T_3 & \cdots & T_{10} & T_{11} \\ T_{12} & T_1 & T_2 & \cdots & T_9 & T_{10} \\ T_{13} & T_{12} & T_1 & \cdots & T_8 & T_9 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ T_{20} & T_{19} & T_{18} & \cdots & T_1 & T_2 \end{bmatrix} \rightarrow \begin{bmatrix} RB_{41} & RB_{11} & RB_{21} & RB_{31} \\ RB_{51} & RB_{42} & RB_{12} & RB_{22} \\ RB_{61} & RB_{52} & RB_{43} & RB_{13} \end{bmatrix} \tag{4}$$

The matrix multiplication is performed group by group. There are $s = m + n - 1 = \lceil L_k/p \rceil + \lceil L/p \rceil$ groups each contains all RBs in a diagonal line. Obviously, each group is constructed with only p elements. The matrix multiplication depends on the order that the p elements contained in the group. For example, the calculation order for the RBs in Fig. 1 is RB_{11}, RB_{12}, RB_{13}, RB_{21}, ..., RB_{61}. The result of multiplying each group with corrected keys is a matrix of one column. The final result is an $M \times 1$ matrix that is the dot product between the result of each group over $GF(2)$, in which the first L_k bits is the final keys. Based on this rule, we can discard the current p elements when performing the next group, which means that we do not need to store all of the reconstituted matrix elements.

By using the RB operation, the theoretical maximum processing speed of the corrected keys is

$$S = \frac{L \cdot f}{(\lceil L/p \rceil + 1)(\lceil L_k/p \rceil)}, \tag{5}$$

where f is the clock frequency of the FPGA. The corresponding maximum secret key rate is given by:

$$\Delta I_m = r \cdot S = \frac{L_k \cdot f}{(\lceil L/p \rceil + 1)(\lceil L_k/p \rceil)}. \tag{6}$$

We have written two programs in MATLAB to verify the correctness of this algorithm, one process data with a conventional matrix multiplication method followed by a mod-2 operation, while the second processes data using the proposed PA algorithm. The calculation results demonstrate that the proposed algorithm is correct.

2.3 Advantages of the Algorithm

There are three advantages in the proposed algorithm:

1) Minimize BRAM usage in the FPGA. The conventional algorithm requires $(1 + r) \times L$ bits of memory to construct the hash functions. In contrast, our algorithm only requires $r \times L$ bits to store the final keys and intermediate bit vectors. Due to the compression ratio r was much lower than one in a long-distance QKD system, the space required for the RAM is smaller than that required in other works.
2) Adapt to different lengths of the corrected and final key. In general, the length of the final key is not a constant due to the variations of the parameters in a QKD system. The proposed PA algorithm adapts to the length of the final key that covered the full range of 0 to L_k bits, as shown in (1), and (2). To ensure the security of a QKD system, the length of the corrected key should be larger than at least a hundred kilobits, the proposed PA algorithm can also adapts to different lengths of the corrected key.
3) Simple data stream read. In the early algorithms based on FPGA [18]–[21], the processing module needs to read elements of the Toeplitz hashing frequently, and the PA process starts after the Toeplitz hashing is generated. However, in the proposed algorithm, the elements only need to be read once, and the PA process proceeds while only p elements of Toeplitz hashing are generated.

3. Hardware Implementation

To completely characterize the PA algorithm, we implemented it in an FPGA using the Verilog language. For ease of implementation, we used the VC707 Evaluation Kit manufactured by Xilinx that includes a Virtex-7 XC7VX485T FPGA and a fixed oscillator with a 200 MHz differential (LVDS)

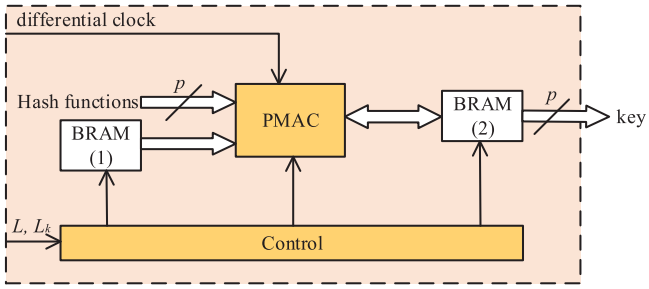


Fig. 2. Logic structure of our PA algorithm.

output. The Virtex-7 FPGA has a total of 303,600 LUTs and 607,200 flip-flops, as well as 2,060 18 kb BRAMs and 1,030 36 kb BRAMs.

A block diagram of the logic structure of our PA algorithm is shown in Fig. 2. BRAM-1 is configured as a true dual port RAM with a width of p and depth of $n + 2$, in which the first and last address space is allocated to store p zeros, and the rest n address spaces are allocated to store the corrected keys. BRAM-2 is configured as a true dual port BRAM with a width of p and depth of m to store the intermediate bit vectors that are updated in each group, and the final secret key in the last update. The capacity of BRAM-2 depends on the length of the final secret keys, which is determined by the parameters of a QKD system [27]. The hash functions are stored in a single port ROM with a width of p and depth of s when testing the algorithm. In the hardware implementation, a logical AND operation is equivalent to a multiplication in $GF(2)$, and the logical exclusion OR (XOR) operation is equivalent to addition in $GF(2)$. We can implement parallel processing of AND and XOR in the PMAC module. The control module is central to the algorithm, and has the following main functions: (1) calculate the total number of rows and columns; (2) generate the data addresses; and (3) control the conversion of rows and columns. In the experiment, the operating differential clock was set to 100 MHz.

The proposed PA algorithm is implemented in three steps. Step 1: depending on the size of Toeplitz hashing, choose a suitable parameter p, where p is the degree of parallelism. In other words, the PMAC module processes p raw key bits each time. Step 2: according to the size of Toeplitz hashing and the degree of parallelism p, calculate the number of rows m and the number of groups s. Step 3: set some reasonable criteria to control the performance of the PA algorithm. When the RBs in the main part are complete and $m_e > m$, jump to the next group, where m_e is the number of rows currently being processed. In the top right corner of the matrix, jump to the next group when the address of the corrected keys is higher than the maximum address. In the bottom left corner of the matrix, p bits of the reconstitution matrix are arranged in reverse order, then jump to the next group when $m_e > m + n - s_e$, where s_e is the number of groups currently being processed.

For any size Toeplitz hashing, it can always be reconstituted as an RB matrix. We can achieve size-adaptive PA algorithm that utilizes the PMAC module to multiply the elements of RB matrix with the corrected keys in FPGA. In principle, there is no upper bound on the size of Toeplitz hashing.

To improve the processing speed, the p elements of the corrected keys are input synchronously without interruption along with the intermediate bit vectors. It takes four clock cycles to process these data. In the first clock cycle, all data needed in the calculation is read. In the second clock cycle, the elements of the hash functions read in the first clock cycle are arranged in reverse order. In the third clock cycle, the PMAC module processes the input corrected keys. In the fourth clock cycle, the result of the PMAC module is written to BRAM-2. In comparison to PA algorithms implemented in software, FPGA-based algorithms use a pipelined design, which can begin processing new data before the prior data processing has finished and, consequently, greatly improve the speed of data processing. A timing diagram is illustrated in Fig. 3 that corresponds to Fig. 1, where h_i and d_j represent the elements of the Toeplitz matrix and corrected keys. The keys km_{ik} correspond to the result of multiplying RB_{ik} with corrected keys, and k_{41}, k_{51}, and k_{61} are the final keys.

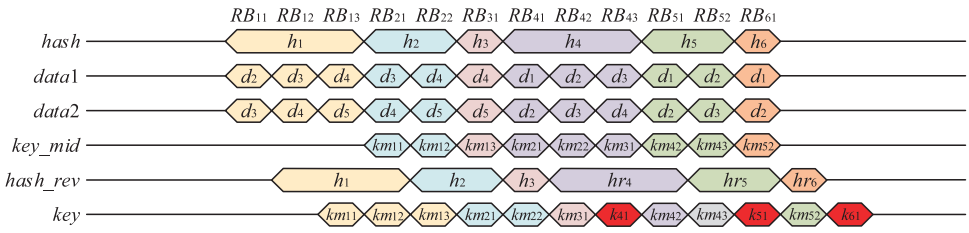


Fig. 3. PA algorithm timing-diagram where *hash* represents the elements of the Toeplitz matrix; *data1* and *data2* represents the corrected keys; *hash_rev* represents the rearranged elements of the Toeplitz matrix; *key_mid* represents the bit vectors of the previous group; and *key* represents the results of the final group.

TABLE 1
Design Space Exploration

PMAC size (bits)	LUTs	Flip-flops	*S* (Mbps)	Simulation Time (ms)	key rate (Mbps)
32 × 32	802 (0.26%)	343 (0.05%)	1.023	0.979	0.1021
64 × 64	2,032 (0.66%)	489 (0.08%)	4.094	0.246	0.4079
128 × 128	7,031 (2.32%)	797 (0.13%)	16.367	0.061	1.6183
256 × 256	26,571 (8.75%)	489 (0.08%)	65.443	0.016	6.4004

We simulated the algorithm using the simulation tool Questa SIM, and the results were consistent with those of MATLAB. After simulation, we implemented the algorithm in hardware on a VC707 evaluation board, and the results of the hardware implementation matched those of the simulations.

4. Results

Before we can determine the size of the PMAC, we must first understand how the speed and area optimizations actually affect the design. For example, the speed of the algorithm can be increased at the expense of additional hardware resources. However, the processing speed of the PA algorithm should not be increased arbitrarily. Instead, the speed should be determined based on the rate of information reconciliation. This approach ensures that the QKD system will consume a minimal number of hardware resources. In Table 1, we presented the design space exploration when the length of input data is 1 Mbits and the compression ratio is 10%. It is shown that the key rate increases linearly with the LUTs. Based on this information, we can choose the optimal size of the PMAC to avoid wasting hardware resources. It is noted that both the data processing speed S and key rate increase linearly with the decreasing length of input data when the compression ratio r remains constant (as shown in (5)). Howerer, when the compression ratio decreases, the data processing speed S grows linearly but key rate keeps unchanged (as shown in (6)).

An overview of several FPGA-based hardware implementations of PA algorithms is shown in Table 2. In [19], four clocks are required to complete each processing unit. While in [20], [21], and the present work, all the clocks of the FPGAs are exploited efficiently due to the adoption of a pipeline architecture algorithm, that means only one clock is required to complete each processing unit. On the other hand, the three algorithms have significant differences in the consumption of RAM. As shown in the Table 2, for a 1 Mbit length of the corrected key, our RAM requirement is only 100 kbit, which is much lower in comparison to [20], [21] where 1095 kbit are required. In [19], the length of the corrected key is 256 kbit, even in this situation, the requirement of our RAM is

TABLE 2
Comparison of Several PA Algorithms

	Zhang *et al.* [19]	Constantin *et al.* [20], [21]	This work
Devices	Altera Cyclone III EP3C120	Xilinx Virtex-6 V6LX240T	Xilinx Virtex-7 XC7VX485T
Length of the corrected key	256,000	995,328	1,000,000
Length of the final key	76,800	0-995,328	0-1,000,000
Processing unit	40×40	512×64	256×256
LUTs	1,902	15,604	26,571
BRAM	656 kb	0kb	100 kb
External RAM	0 kb	1,095 kb	0 kb
Clock frequency	40 MHz	125 MHz	100 MHz
Max. key rate	0.07 Mbps	4.1 Mbps	6.4 Mbps

still much less than this work where 656 kbit are occupied ([18] does not provides enough detailed parameters, so we do not list it here).

In above, we have shown that the rhomboid-block operations can reduce the consumption of RAM from 1.1 Mbit to 100 kbit when the block size is 1 Mbit and compression ratio is 10%. Although 1 Mbit is not a large resource for a FPGA, this is not the case for a large input data block. Due to the finite size effect of the QKD, large block size is necessary to ensure a long distribution distance and high secret key rate [22], [23]. For a typical block size of 100 Mbit, the occupation of RAM will increase to 110 Mbit, which is beyond the memory capacity of the on-chip RAM for a Xilinx Virtex-7 FPGA which has a typical value of several tens of Mbit. In this case, a PA FPGA implementation with a low memory storage requirement is crucial. By using our approach, the occupation of RAM resource can decrease to 10 Mbit, which is within the memory capacity of the on-chip RAM of the FPGA, In addition, the control of data stream is more simple in accordance with our approach. Therefore, our work is beneficial to the FPGA-based implementation of the PA in practical QKD systems where a large data block size is required.

5. Conclusions and Outlook

This article provides a detailed description of our high-speed PA algorithm, and presents the verification results from an FPGA implementation. Compared to other works, the proposed PA algorithm requires less FPGA BRAM and allows fine adjustment of the lengths of input and output blocks so that it can meet the requirements of different QKD systems. In the future, we will implement information reconciliation [28] in an FPGA, and develop a complete practical secret key distillation engine.

References

[1] C. H. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," in *Proc. IEEE Int. Conf. Comput., Syst. Signal Process.*, Bangalore, India, 1984, pp. 175–179.
[2] N. Gisin, G. G. Ribordy, W. Tittel, and H. Zbinden, "Quantum cryptography," *Rev. Mod. Phys.*, vol. 74, no. 1, pp. 145–195, Jan. 2002.
[3] S. L. Braunstein and P. van Loock, "Quantum information with continuous variables," *Rev. Mod. Phys.*, vol. 77, no. 2, pp. 513–577, Jun. 2005.
[4] X.-B. Wang, T. Hiroshima, A. Tomita, and M. Hayashi, "Quantum information with Gaussian states," *Phys. Rep.*, vol. 448, no. 1–4, pp. 1–111, Aug. 2007.

[5] V. Scarani, H. Bechmann-Pasquinucci, N. J. Cerf, M. Dusek, N. Lutkenhaus, and M. Peev, "The security of practical quantum key distribution," *Rev. Mod. Phys.*, vol. 81, no. 3, pp. 1301–1350, Sep. 2009.
[6] C. Weedbrook *et al.*, "Gaussian quantum information," *Rev. Mod. Phys.*, vol. 84, no. 2, pp. 621–669, May. 2012.
[7] H.-K. Lo, M. Curty, and K. Tamaki, "Secure quantum key distribution," *Nature Photon.*, vol. 8, no. 8, pp. 595–604, Jul. 2014.
[8] E. Diamanti and A. Leverrier, "Distributing secret keys with Quantum continuous variables: Principle, security and implementations," *Entropy*, vol. 17, no. 9, pp. 6072–6092, Aug. 2017.
[9] Y.-M. Li, X.-Y. Wang, Z.-L. Bai, W.-Y. Liu, S.-S. Yang, and K.-C. Peng, "Continuous variable quantum key distribution," *Chin. Phys. B*, vol. 26, no. 4, Apr. 2017, Art. no. 040303.
[10] C. H. Bennett, G. Brassard, and J.-M. Robert, "Privacy amplification by public discussion," *SIAM J. Comput.*, vol. 17, no. 2, pp. 210–229, 1988.
[11] C. H. Bennett, G. Brassard, C. Crepeau, and U. M. Maurer, "Generalized privacy amplification," *IEEE Trans. Inf. Theory*, vol. 41, no. 6, pp. 1915–1923, Nov. 1995.
[12] G. Van Assche, *Quantum Cryptography and Secret-Key Distillation.* Cambridge, U.K.: Cambridge Univ. Press, 2006.
[13] C.-M. Zhang *et al.*, "Fast implementation of length-adaptive privacy amplification in quantum key distribution," *Chin. Phys. B*, vol. 23, no. 9, Sep. 2014, Art. no. 090310.
[14] D. Zou and I. B. Djordjevic, "FPGA-based rate-compatible LDPC codes for the next generation of optical transmission systems," *IEEE Photon. J.*, vol. 8, no. 5, Oct. 2016, Art. no. 7906008.
[15] S. Keskin and T. Kocak, "GPU-based gigabit LDPC decoder," *IEEE Commun. Lett.*, vol. 21, no. 8, pp. 1703–1706, Aug. 2017.
[16] R. Takahashi, Y. Tanizawa, and A. R. Dixon, "High-speed implementation of privacy amplification in quantum key distribution," presented at 6th Int. Conf. Quantum Cryptography, Washington, DC, USA, 2016, Paper 160.
[17] P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "A survey of FPGA-based LDPC decoders," *IEEE Commun. Surveys Tut.*, vol. 18, no. 2, pp. 1098–1122, May. 2016.
[18] A. Tanaka *et al.*, "High-speed quantum key distribution system for 1-Mbps real-time key generation," *IEEE J. Quantum Electron.*, vol. 48, no. 4, pp. 542–550, Apr. 2012.
[19] H.-F. Zhang *et al.*, "A real-time QKD system based on FPGA," *J. Lightwave Technol.*, vol. 30, no. 20, pp. 3226–3234, Oct. 2012.
[20] J. Constantin *et al.*, "An FPGA-based 4 Mbps secret key distillation engine for quantum key distribution systems," *J. Sign. Process. Syst.*, vol. 86, no. 1, pp. 1–15, Jan. 2017.
[21] N. Walenta *et al.*, "A fast and versatile quantum key distribution system with hardware key distillation and wavelength multiplexing," *New J. Phys.*, vol. 16, no. 1, Jan. 2014, Art. no. 013047.
[22] V. Scarani and R. Renner, "Quantum cryptography with finite resources: Unconditional security bound for discrete-variable protocols with one-way postprocessing," *Phys. Rev. Lett.*, vol. 100, no. 20, May 2008, Art. no. 200501.
[23] M. Lucamarini *et al.*, "Efficient decoy-state quantum key distribution with quantified security," *Opt. Express*, vol. 21, no. 21, pp. 24550–24565, Oct. 2013.
[24] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," *J. Comput. Syst. Sci.*, vol. 18, pp. 143–154, Apr. 1979.
[25] H. Krawczyk, "LFSR-based hashing and authentication," in *Proc. 14th Annu. Int. Cryptol. Conf.*, Santa Barbara, CA, USA, 1994, pp. 129–139.
[26] P. Eraerds, N. Walenta, M. Legre, N. Gisin, and H. Zbinden, "Quantum key distribution and 1 Gbps data encryption over a single fibre," *New J. Phys.*, vol. 12, no. 6, Jun. 2010, Art. no. 063027.
[27] X.-L. Sun, I. B. Djordjevic, and M. A. Neifeld, "Secret key rates and optimization of BB84 and decoy state protocols over time-varying free-space optical channels," *IEEE Photon. J.*, vol. 8, no. 3, Jun. 2016, Art. no. 7904713.
[28] Z.-L. Bai, S.-S. Yang, and Y.-M. Li, "High-efficiency reconciliation for continuous variable quantum key distribution," *Jpn. J. Appl. Phys.*, vol. 56, no. 4, Apr. 2017, Art. no. 044401.