Postprocessing for quantum random-number generators: Entropy evaluation and randomness extraction

Xiongfeng Ma,[1,2,*] Feihu Xu,[2,†] He Xu,[2] Xiaoqing Tan,[2,3,‡] Bing Qi,[2,§] and Hoi-Kwong Lo[2,‖]

[1]*Center for Quantum Information, Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China*

[2]*Center for Quantum Information and Quantum Control, Department of Physics and Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ontario, Canada*

[3]*Department of Mathematics, College of Information Science and Technology, Jinan University, Guangzhou, Guangdong, China*

(Received 24 April 2013; published 21 June 2013)

Quantum random-number generators (QRNGs) can offer a means to generate information-theoretically provable random numbers, in principle. In practice, unfortunately, the quantum randomness is inevitably mixed with classical randomness due to classical noises. To distill this quantum randomness, one needs to quantify the randomness of the source and apply a randomness extractor. Here, we propose a generic framework for evaluating quantum randomness of real-life QRNGs by min-entropy, and apply it to two different existing quantum random-number systems in the literature. Moreover, we provide a guideline of QRNG data postprocessing for which we implement two information-theoretically provable randomness extractors: Toeplitz-hashing extractor and Trevisan's extractor.

DOI: 10.1103/PhysRevA.87.062327 PACS number(s): 03.67.Dd, 42.50.Ex

I. INTRODUCTION

Random numbers play a crucial role in many fields of science, technology, and industry—for instance, cryptography, statistics, scientific simulations [1], and lottery. Pseudorandom-number generators (pseudo-RNGs) based on computational complexities have been well developed in the past few decades [2] and can generate high-speed random numbers with little cost. However, the main drawback of pseudo-RNGs is that the generated randomness is not information-theoretically provable. In fact, all of the (software-based) pseudo-RNGs can be realized by a deterministic algorithm given sufficient computational power. This pseudorandomness would cause problems in many applications, such as those in cryptography [3,4]. Recently, Microsoft confirms that XP contains RNG bugs;[1] security flaws have been found in online encryption methods due to imperfections of random-number generation.[2]

To address the security issues introduced by pseudo-RNGs, physical RNGs have been developed [5–7]. In particular, the probabilistic nature of quantum mechanics offers a natural way to build an information-theoretically provable RNG [5]—quantum random-number generators (QRNGs). Note that some physical RNGs have been included in microprocessors,[3] although the generated randomness is not quantum mechanical in nature.[4]

In theory, a QRNG can produce random numbers with provable randomness. In practice, quantum signals (the source of true randomness[5]) are inevitably mixed with classical noises. An adversary (Eve) can, in principle, control the classical noise and gain partial information about the raw random numbers. In this work, we assume a trusted device scenario, but the classical noise might be deterministic if we calibrate the system more carefully. For instance, imagine that an input to our device is an *external* power supply. Imagine further that through carefully monitoring the input value of the power to our device, we have determined that the power may still fluctuate by, say, up to 1%. In principle, the source of such fluctuations of an external power supply might be the action of an adversary—Eve—who, therefore, has complete knowledge about the actual value of the power supply at all times.

Therefore, it is necessary to apply a postprocessing procedure to distill out the true randomness that Eve has *almost*

*xma@tsinghua.edu.cn
†feihu.xu@utoronto.ca
‡ttanxq@jnu.edu.cn
§bqi@physics.utoronto.ca
‖hklo@comm.utoronto.ca

[1]See the news from the Computerworld, "Microsoft confirms that XP contains random number generator bug," Gregg Keizer, November 21, 2007.

[2]See the news from the New York Times, "Flaw Found in an Online Encryption Method," John Markoff, February 14, 2012.

[3]See, for example, "Intel Corporation Intel 810 Chipset Design Guide," June 1999, Ch. 1.3.5, pages 1–10, download.intel.com/design/chipsets/designex/29065701.pdf.

[4]"Evaluation of VIA C3 'Nehemiah' Random Number Generator," Cryptography Research, Inc., February 2003, www.cryptography.com/public/pdf/VIA_rng.pdf.

[5]We define "true randomness" when the randomness is information-theoretically provable. One of the main objectives of our work is to give a randomness extraction procedure such that the extracted output numbers are proven random. On one hand, quantum mechanics comes with randomness in nature. For example, no one can predict the results of the vacuum fluctuation. On the other hand, it is not clear whether or not one can extract "true" randomness from the classical noises. That is, we have not proved the randomness extracted from the classical noises. Thus, from a conservative point of view (especially for the usage in cryptography), we only extract random numbers that are information-theoretically provable.

no information about. This distilling procedure is called *randomness extraction*, realized by employing *randomness extractors*. In other words, randomness extractors are used for distilling the true randomness and eliminating the effect of classical noises. The goal of randomness extractors [8] is to extract (almost) perfect randomness from the raw data generated from a practical QRNG with the help of a short random seed, which requires an extra source of randomness. The key input parameter of a randomness extractor is the min-entropy (see Definition I.1) of raw data. Nonetheless, a general method to quantify the min-entropy from the raw data of a QRNG is still missing.

For randomness extraction, previously, some simple methods have been widely used for QRNGs. For instance, an exclusive-OR (XOR) operation has been employed in the literature [9,10]: dividing the raw data into two bit strings and performing a bitwise XOR operation between them. In addition, a least-significant-bits operation [6,7] or nonuniversal hashing functions [11] have been proposed and implemented for QRNGs. These operations can certainly refine the raw data to pass some randomness statistical tests. However, the key point is, the generated randomness is not information-theoretically provable. Recently, a more sophisticated randomness extraction procedure is proposed in Ref. [12], which quantifies the randomness by Shannon entropy instead of min-entropy and applies nonuniversal hash functions for extraction. Unfortunately, the randomness extracted there is still not information-theoretically provable due to the following two reasons: randomness cannot be well quantified by Shannon entropy [13,14] and the randomness from nonuniversal hashing functions relies on computational assumptions.

In contrast, an important and promising randomness extractor, Trevisan's extractor [15,16], raised considerable theoretical interest not only because of its data parsimony compared to other constructions, but particularly because it is secure against quantum adversaries [17]. The seed length of Trevisan's extractor is polylogarithmic in the length of the input and it can also be proven to be a strong extractor (see Theorem 22 in [18]). That is, the random seed of Trevisan's extractor can be reused. This is particularly important since for a popular universal-hashing function such as Toeplitz hashing [19,20], which has been well developed for privacy amplification [21] in quantum key distribution (QKD)[6]—the random seed used to construct a Toeplitz matrix is longer than the output string. This means that *no* net randomness can be extracted if the universal hashing is directly used for randomness extraction. Despite the considerable theoretical attention of Trevisan's extractor, its practical implementation is still missing. This is probably because Trevisan's extractor has a rather complex structure that the quantum information community may not be familiar with and its implementation involves the nontrivial tradeoff between speed and the values of security parameters.

Here, we fill the above two gaps: a general method for the quantification of randomness and a practical implementation of Trevisan's extractor. We report a generic scheme that can process the raw data from a QRNG to random numbers that (nearly) follow a uniform distribution. The two main contributions of this work can be stated as follows:

(1) We present a framework for quantifying the quantum randomness from QRNG by min-entropy and discuss how one can evaluate this min-entropy in a physical device. We apply our framework to two different existing QRNGs in the literature [12,22].

(2) We provide prototypical implementation of Trevisan's extractor and show how such implementation can be used in real-life QRNGs. From this implementation, we discover the major computational step that limits Trevisan's extractor speed.

Based on a few reasonable assumptions on the physical model of QRNG, the randomness extracted from the proposed postprocessing is information-theoretically provable. Our generic postprocessing scheme consists of three steps: (a) model and characterize the QRNG setup through measurements; (b) quantify the quantum randomness of the raw data with min-entropy; and (c) apply a randomness extractor. To illustrate the generality of our method, our scheme is applied to *two* different types of real-life QRNGs [12,22].

Besides attempt of Trevisan's extractor, we also implement the Toeplitz-hashing extractor and prove that the universal-hashing function [23] constructs a strong extractor (see Definition I.4), and thus allows us to maintain the randomness of the seed for subsequent applications. The implementation speeds for Trevisan's and the Toeplitz-hashing extractors are 0.7 and 441 kb/s, respectively. The outcomes from both extractors pass all the standard randomness tests we exploited.

Compared to the Toeplitz-hashing, Trevisan's extractor requires a seed with a shorter length. Thus, Trevisan's extractor has a certain advantage in cases where the random seed bits are limited. We note that our prototypical implementation of Trevisan's extractor allows researchers to better understand the complexity and difficulty in the implementation of Trevisan's extractor, thus paving the way to future implementations. Recently, Maurer *et al.* posted a follow-up paper on our work in which they built on our results and improved the speed of implementation [24].

The rest of this paper is organized as follows. We introduce related notations and definitions in the rest of this section. In Sec. II, we present a procedure to evaluate the min-entropy of the quantum signals. We implement Trevisan's extractor in Sec. III A and a universal hashing (Toeplitz-hashing) extractor in Sec. III B. In Sec. IV, we show the results of statistical tests. Finally, we conclude this paper in Sec. V.

A. Notations and definitions

Notations. U_d represents a uniform distribution on $\{0,1\}^d$. The outcome of an *ideal* RNG, described by a random variable, follows a uniform distribution.

[6]Randomness extractors can also be used for privacy amplification [21] in quantum key distribution (QKD). Note that privacy amplification is a crucial step in QKD postprocessing. A few randomness extractors have been proven to be secure against quantum side channels [17]. The main advantage is that no (or little) classical communication is required for privacy amplification. It is an interesting prospective research topic to apply the techniques developed in randomness extraction for privacy amplification.

Min-entropy is widely used for quantifying the randomness of a probability distribution [13,14].

Definition I.1 (Min-entropy). The min-entropy of a probability distribution X on $\{0,1\}^n$ is defined by

$$H_\infty(X) = -\log_2\left(\max_{v\in\{0,1\}^n} \text{Prob}[X=v]\right). \quad (1)$$

In cryptography, the deviation of a practical protocol from an ideal protocol is characterized by a *security parameter*, ε. The statistical distance is commonly used as a standard security measure.

Definition I.2 (ε-close). Two probability distributions X and Y over the same domain T are ε-close if the statistical distance between them is bounded by ε,

$$\|X-Y\| \equiv \max_{V\subseteq T}\left|\sum_{v\in V}(\text{Prob}[X=v]-\text{Prob}[Y=v])\right| = \frac{1}{2}\sum_{v\in T}|\text{Prob}[X=v]-\text{Prob}[Y=v]| \leqslant \varepsilon. \quad (2)$$

Here, the second equality in Eq. (2) can be proven by induction on the domain size $|T|$. It is obvious that the equality holds for $|T| = 1,2$. For the case of $|T| \geqslant 3$, one can always find two v_1 and v_2 in T such that $(\text{Prob}[X=v_1]-\text{Prob}[Y=v_1])(\text{Prob}[X=v_2]-\text{Prob}[Y=v_2]) \geqslant 0$. Then, one can combine v_1 and v_2 together as a new variable v' to form a new domain T'. If the statement of the equality $S(|T|)$ holds, the above argument shows that $S(|T|+1)$ also holds. Since the statement holds for $S(1)$ and $S(2)$, by induction, it holds for all $S(|T|)$ with $|T| \geqslant 1$.

Roughly speaking, the statistical distance quantifies the distinguishability of two probability distributions. The factor of $1/2$ in Eq. (2) is used to normalize the statistical distance so that its value falls into $[0,1]$. When X is ε-close to Y, X is indistinguishable from Y except for a small probability ε. For example, the output of a practical RNG is said to be ε-close to an ideal RNG if it satisfies Definition (2). We emphasize that the security parameters from Definition (2) are composable. The notion of composability was first proposed in the classical cryptography for the study of security when composing classical cryptographic protocols in a complex manner [25,26]. It is introduced to quantum cryptography by Refs. [27,28].

Definition I.3 (Extractor). A (k,ε,n,d,m)-extractor is a function

$$\text{Ext}: \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m, \quad (3)$$

such that for every probability distribution X on $\{0,1\}^n$ with $H_\infty(X) \geqslant k$, the probability distribution $\text{Ext}(X,U_d)$ is ε-close to the uniform distribution on $\{0,1\}^m$.

In short, an extractor is a function that takes a small seed of d bits and a partially random source of n bits to output an almost perfect random bit string of m bits.

Definition I.4 (Strong extractor). A (k,ε,n,d,m)-strong extractor $\text{Ext}(X,U_d)$ is an extractor such that the probability distribution $\text{Ext}(X,U_d)\circ U_d$ is ε-close to the uniform distribution on $\{0,1\}^{m+d}$.

Note that the key advantage of a strong extractor is that the input (random) seed can be reused (with a security parameter increased by ε). Thus, one can partition the output of a practical RNG into (small) blocks and process them by a strong extractor with the same seed.

Definition I.5 (Universal hashing). A family of hash functions $\mathcal{H}$, mapping S to T, is two-universal if

$$\text{Prob}_{h\in\mathcal{H}}\{h(x)=h(y)\} \leqslant \frac{1}{|T|}, \quad (4)$$

for all $x \neq y \in S$.

II. QUANTUM RANDOMNESS EVALUATION

In this section, we provide a general framework to evaluate the quantum (true) randomness from a practical QRNG. The QRNGs developed in Refs. [12,22] are discussed as illustrations of the evaluation process. We note that our evaluation procedure is a generic method that can be applied to other QRNGs with certain modifications.

A. Physical model

In general, the random numbers of a QRNG come from a certain measurement. We refer to the measurement outcome as *quantum signal*. This quantum signal is inevitably mixed with *classical noises*, such as background detections and electronic noises. From a cryptographic view, these classical noises might be known to (or even manipulated by) Eve in the worst case scenario. Hence, the main objective of the postprocessing for a QRNG is to extract out the quantum (true) randomness and eliminate the contributions of classical noises.

Let us consider a generic flow chart of QRNG, as shown in Fig. 1. First, a quantum state is prepared, which is the source of true randomness. Then, a measurement is performed on the quantum state. Finally, the raw data is postprocessed by a randomness extractor to generate truly random numbers. For example, the quantum state in Ref. [22], which characterizes the random phases of the photons from spontaneous emissions, is prepared by operating a laser near its threshold level; the measurement is operated by a delayed self-heterodyning system; and the raw data is evaluated based on a physical model and processed by randomness extractors. For the QRNG in Ref. [12], the quantum state is produced by the quadrature amplitude of the vacuum state;

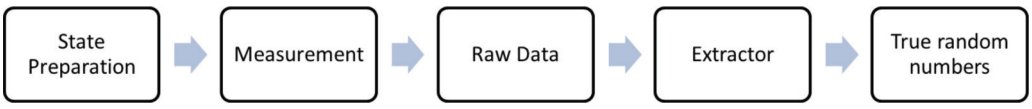

FIG. 1. (Color online) A generic schematic diagram of a QRNG setup. A quantum state is prepared and measured. Then the outcome is processed to generate random numbers by an extractor.

the measurement is realized by a homodyne detector; and the raw data is processed by a hash extractor.

B. Quantum randomness evaluation

The key parameter we need to evaluate here is the min-entropy, defined in Eq. (1), of the quantum signal contained in the raw data. In the following, we present a method to evaluate the min-entropy by deriving the probability distribution of the quantum signal.

Let us take the QRNG setup in Refs. [10,22] as the first example to show the detailed procedures of the evaluation process. In this case, the quantum signal comes from vacuum fluctuation and the contribution from all other sources of phase fluctuations is defined as classical noise. The details of this QRNG can be found in Ref. [22]. The assumptions of the physical model needed for the derivation of the probability distribution of the quantum signal are listed as follows:

(1) The total signal is a mixture of quantum signal and classical noise. Quantum signal is independent of classical noise.

(2) The quantum signal follows a Gaussian distribution. The total analog signal is digitalized by an analog-to-digital convertor (ADC).

(3) The ratio between the variances of quantum signal and classical noise can be determined, denoted by γ.

(4) Total signal variance can be characterized by sampling, denoted by σ^2_{total}. Note that the last assumption can be satisfied when the sequence of the raw data is independent and identically distributed (i.i.d.). Also, there is no need to assume that classical noise follows a Gaussian distribution.

In summary, to derive the probability distribution of the quantum signal, the key point is to find out its variance. This is done by measuring the total variance of the raw data and the quantum-to-classical ratio. That is, with assumptions (1), (3), and (4), one can easily derive the quantum variance,

$$\sigma^2_{\text{quantum}} = \frac{\gamma\sigma^2_{\text{total}}}{1+\gamma}. \tag{5}$$

From the variance together with the Gaussian distribution assumption, one can get the whole probability distribution of the quantum signal. Since the analog signal is sampled by an eight-bit ADC to generate digital bits [22], one can evaluate the probability distribution of the digitalized output on $\{0,1\}^8$, given the Gaussian distribution of the quantum signal. Then the min-entropy of the quantum signal can be derived by Definition I.1. Following the detailed calculation procedures in Ref. [22], a min-entropy of 6.7 bits per eight-bit raw sample (from an eight-bit ADC) is obtained. In Ref. [22], the authors use the ADC to cut off bins evenly along the voltage axis. By carefully designing the bin size, one might increase the min-entropy.

Next, we discuss how our evaluation process can be applied to Ref. [12]. The assumptions of a physical model are similar to the ones discuss above in that the vacuum state fluctuation used there [12] also follows a Gaussian distribution. To rigorously quantify the randomness by min-entropy, from the data about the variance of total signal and classical noise there (see Fig. 2 in Ref. [12]), the variance of quantum signal can be derived. Afterward, instead of calculating Shannon entropy there, the min-entropy of the quantum signal can be derived by Definition I.1 given the Gaussian distribution of the quantum signal about the probability distribution of the digitalized output on $\{0,1\}^{16}$.[7]

We note that our framework can also be applied to other types of QRNGs, such as those with discrete variables rather than continuous variables. The key point is to evaluate the min-entropy of quantum signals through quantitatively separating its contributions from quantum signals and classical noises. Such an extension would be an interesting prospective research topic.

C. Upper bound of randomness

The randomness of a given QRNG setup is a limited resource. This can be shown by providing the upper bound of randomness, say, via Shannon entropy, that one can extract from the measurement outcome.[8] The upper bound also indicates how much margin is left for further improvement in postprocessing.

Here, since we only have the experimental data of Ref. [22], we take it as an example to show how one can evaluate the upper bound of entropy for a practical QRNG setup. The quantum signal is measured by a photodetector (PD). Given a perfect photon-number resolving detector, the upper bound of the min-entropy is determined by the photon number within the detection time window. The laser power used in the setup is 0.95 mW,[9] which corresponds to 1.5×10^6 photons at 1550 nm within a 200 ps detection time window. Thus, the maximal entropy of a sample from the PD can be estimated by $\log_2(1.5 \times 10^6) = 20.5$ bits, which is the upper bound of the min-entropy of the QRNG source.

III. RANDOMNESS EXTRACTION

In this section, we will present our prototypical implementations of Trevisan's extractor and the Toeplitz-hashing extractor. These implementations can be easily used for the extraction of Refs. [12,22], and other more general QRNGs.

A. Trevisan's extractor

1. *Results summary*

Trevisan proposed an approach to construct randomness extractors based on pseudorandom-number generators [15]. Trevisan's extractor has a number of important theoretical

[7]Without the detailed data values of Ref. [12], we did not calculate the final result of min-entropy. As indicated earlier in Sec. II B, the missing experimental parameters are the total variance of quantum signals and classical noises and the ratio between the variances of quantum signals and classical noises.

[8]Roughly speaking, the min-entropy can be regarded as the lower bound of randomness one can extract, whereas Shannon entropy can be treated as the upper bound. The min-entropy is always no greater than the corresponding Shannon entropy.

[9]In principle, one can go beyond this limitation by increasing the laser power. However, the upper bound of min-entropy can only increase logarithmically with power intensity.

advantages.[10] First, it is secure against a quantum adversary. Second, the seed length is polylogarithmic in the length of the input. Third, it can also be proven to be a strong extractor with certain modifications on the security parameters (Theorem 22 in [18]). However, a real-life implementation of this important extractor was never reported in the literature.

Here, we implement its improved version by Raz, Reingold, and Vadhan [16]. There are two main steps to construct Trevisan's extractor: one-bit extractor and combinatorial design. The one-bit extractor can be realized by an error correcting code, which is constructed by concatenating a Reed-Solomon code with a Hadamard code, as shown in Appendix A of Ref. [18]. For the combinatorial design part, we implement a refined version of Nisan-Wigderson design [2,29].

Our implementation of Trevisan's extractor yields an output speed of 0.7 kb/s. From our result, we locate the major computational step that limits Trevisan's extractor speed, which lies on the error-correction-based one-bit extractor. Recently, Maurer *et al.* posted a follow-up paper [24] on our work, where alternative one-bit extractors, rather than listing errorcorrecting codes, are utilized. As a result, the extraction speed in the implementation of Maurer *et al.* is substantially improved [24], which can go up to 150 kb/s. Compared to the Toeplitz-hashing extractor (Sec. III B), Trevisan's extractor requires a seed with a shorter length. Thus, Trevisan's extractor has a certain advantage in cases where the random seed bits are limited. Although our implementation results show that the speed of the Toeplitz-hashing extractor (441 kb/s) is much faster than that of Trevisan's extractor (0.7 kb/s), a detailed comparison between these two extractors is an interesting prospective research project.

2. *Implementation procedure*

We sketch out the implementation procedure in this section, while the implementation details are shown in Appendix A. We input an n_i-bit string, raw data from a QRNG, with a min-entropy at least k, and a d-bit random seed (y) into an $(k,\varepsilon,n_i,d,n_f)$ extractor, constructed by combining an $(n_i,1/2-\varepsilon/4n_f)$ error correcting code[11] and an (m_e,ρ) design,[12] and then output an n_f-bit string which is ε-close to a uniform distribution. Here, k, ε, and n_i are given from the source and practical use of random numbers. We need to figure out d (as small as possible) and n_f (as large as possible).

(1) Map the input n_i-bit string to an $\bar{n}$-bit string according to the $(n_i,1/2-\varepsilon/4n_f)$ error correcting code. Here, $\bar{n}$ can be assumed to be a power of 2 [15]. In practice, one can concatenate the Reed-Solomon code and Hadamard code together (see Appendix A of [18]), where the codeword length is given by

$$\bar{n}=2^{2m_e},\quad m_e=\lceil \log_2 n_i+2\log_2 n_f-2\log_2\varepsilon+4\rceil. \tag{6}$$

Also, n_f can be upper bounded by k for the error correcting code construction.

(2) Construct an (m_e,ρ) design [29], with

$$\begin{aligned} m_e &= \tfrac{1}{2}\log_2\bar{n}=O(\log_2(n_i/\varepsilon)),\\ \rho &= [k-3\log_2(n_f/\varepsilon)-d-3]/n_f, \end{aligned} \tag{7}$$

where the second equation is from Proposition 10 and Theorem 22 in Ref. [18], typically $1\leqslant\rho\leqslant 1.5$. The design parameter ρ can be viewed as the ratio of min-entropy that can be extracted. One can simply pick up $\rho=1$ if the output length is to be optimized (Lemma 17 in Ref. [18]). The extractor seed, with a length of d, is composed of blocks of seeds with lengths of the square of the smallest power of 2 which is greater than m_e. Note that this block design idea is proposed by Raz *et al.* [18]. Here, we are interested in a design with $\rho=1$, so that most of randomness can be extracted. According to the explicit design proposed by Nisan and Wigderson [2] and proved in Refs. [29,30], the number of such blocks and hence the seed length are given by

$$\begin{aligned} b &= \lceil\log_2 n_f\rceil-m_d+1,\\ d &= 2^{2m_d}b=O(\log_2^2(n_i/\varepsilon)\log_2 n_i),\\ m_d &\equiv \lceil\log_2 2m_e\rceil. \end{aligned} \tag{8}$$

In fact, any design with $d\geqslant\lceil\log_2\bar{n}\rceil^2 b$ and $\rho=1$ can be applied here.

(3) The ith bit of the n_f-bit output is given by the y_{S_i}th bit of the encoded $\bar{n}$-bit string, where y_{S_i} is a substring of y, formed by the bits of y at the positions given by the elements of S_i.

B. Universal hashing

Owing to the similarity between the definitions of extractors and privacy amplification [21], any privacy amplification scheme can be used as an extractor, in principle. However, there is one subtle difference. In privacy amplification, the random seed (public randomness) is assumed to be free, whereas in an extractor, one needs to take the seed into account as it does consume random bits. Therefore, a direct transplant of privacy amplification schemes may not work for randomness extraction. In fact, for a popular universal hashing function, Toeplitz hashing [19,20], the random seed used to construct a Toeplitz matrix is longer than the output string. This means that no net randomness can be extracted if the universal hashing is directly used for randomness extraction. To overcome this problem, one needs to prove that the privacy amplification scheme constructs a strong extractor (see Definition I.4), thus allowing one to maintain the randomness of the seed for subsequent applications. Fortunately, the extractors constructed by universal hashing functions [23] (see Definition I.5) can be easily proven to be strong extractors by the Leftover Hash Lemma [8].

Lemma III.1 (Leftover Hash Lemma [8]). Let $\mathcal{H}=\{h_1,h_2,\ldots,h_{2^d}\}$ be a (two-)universal hashing family, mapping from $\{0,1\}^n$ to $\{0,1\}^m$, and X be a probability distribution on $\{0,1\}^n$ with $H_\infty(X)\geqslant k$. Then for $x\in X$ and $h_y\in\mathcal{H}$ where $y\in U_d$, the probability distribution formed by $h_y(x)\circ y$ is $\varepsilon=2^{(m-k)/2}$ close to U_{m+d}. That is, it forms a $(k,2^{(m-k)/2},n,d,m)$-strong extractor.

We note that Lemma III.1 also implies that the Toeplitz matrix may be reused in the privacy amplification of

[10] For a discussion, see, for example, a follow-up paper by Mauerer *et al.* [24].

[11] An (n,p) error correcting code has a codeword length of n and is able to correct error rates up to p.

[12] An (m,ρ) design means a collection of m subsets. The average number of overlap elements between two subsets is no more than ρ. Details can be found in Ref. [29].

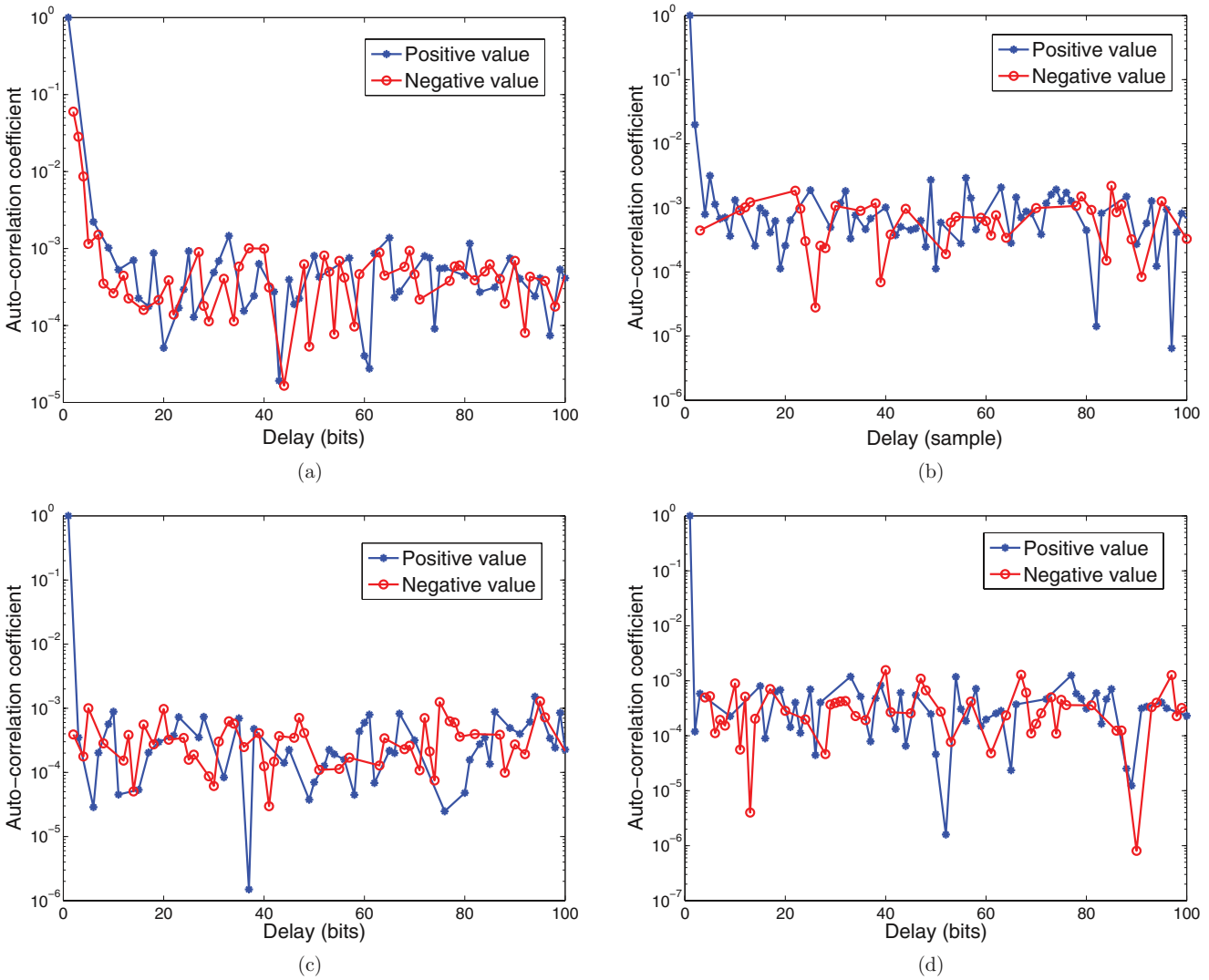

FIG. 2. (Color online) Autocorrelation evaluation results. All normalized correlation is evaluated from a 10 Mb record of the raw data. (a) Autocorrelation of the raw data (between bits). The average value of autocorrelation coefficient is 9.5×10^{-4}. The most significant correlations are within eight bits, due to the usage of eight-bit ADC. (b) Autocorrelation of the raw data (between samples). The average value is 4.9×10^{-4}. (c) Autocorrelation of the outcomes from the Toeplitz-hashing extractor. The average value is -1.0×10^{-5}. (d) Autocorrelation of the outcomes from Trevisan's extractor. The average value is 1.6×10^{-5}. In theory, for a truly random 10×10^6 bit string, the average normalized correlation coefficient is 0 with a standard deviation of 4×10^{-4}.

QKD. Then, one can use a private key (as a seed) to construct a Toeplitz matrix for privacy amplification without compromising (much of) the privacy of the seed. Hence, reusing the seed can save the classical communication for privacy amplification, which is normally required in standard QKD postprocessing [31]. It is also practically beneficial for privacy amplification to divide the raw key data into small blocks and apply a small Toeplitz matrix individually. However, the finite-size effect of a small block can significantly lower the privacy amplification efficiency [32]. This issue is an interesting research topic for future study.

Here, we use Toeplitz matrices for universal hashing function construction [19,20] and implement the Toeplitz-hashing extractor. A Toeplitz matrix of dimension $n \times m$ requires only the specification of the first row and the first column, and the other elements of the matrix are determined by descending diagonally down from left to right. Thus, the total number of random bits required to construct (choose) a Toeplitz matrix is $n + m - 1$.

The procedure of Toeplitz-hashing extractor is given as follows:

(1) Given raw data of size n with a min-entropy of k and a security parameter ε, determine the output length to be

$$m = k - 2 \log_2 \varepsilon. \tag{9}$$

(2) Construct a Toeplitz matrix with an $n + m - 1$-random-bit seed.[13]

(3) The extracted random-bit string is obtained by multiplying the raw data with the Toeplitz matrix.

[13]For demonstration purposes, we use pseudorandom numbers for this step.

We implement a Toeplitz-hashing extractor to the QRNG presented in Ref. [22]. As mentioned in Sec. II, the min-entropy of the raw data is bounded by 6.7 bits per sample. With the input bit-string length of $2^{12} = 4096$, the output bit-string length is $4096 \times 6.7/8 \geqslant 3430$. Thus, we use a 4096×3230 Toeplitz matrix for randomness extraction, which results in $\varepsilon < 2^{-100}$ as from Eq. (9). Our implementation of a Toeplitz-hashing extractor achieves generation rates of 441 kbits/s.[14]

Notice that recently Toeplitz matrix hashing is implemented for QKD privacy amplification with a block size exceeding 10^6 [33]. As discussed above, privacy amplification requires a big block size due to the finite-size key effect [31], whereas in the application of randomness extraction, a small block size will only reduce the efficiency. Nevertheless, the technique developed in [33] could be useful for extractor implementations as well, which we will leave for future investigation.

IV. RANDOMNESS TEST

A. Statistical tests

We apply three standard statistical tests—DIEHARD,[15] NIST,[16] and TESTU01 [34]—to evaluate our results. Again, since we only have the data about the QRNG of Ref. [22], we use it as the test of our implementation of randomness extractors. First, the raw data from the QRNG [22] does not pass the statistical tests due to the classical noises mixed in the raw data and the fact that the as-obtained quantum signals follow a Gaussian distribution instead of a uniform distribution. Secondly, the random numbers from a pseudo-RNG cannot pass all the tests, which exposes its underlying determinism. Finally, we repeatedly operate the Toeplitz-hashing extractor and Trevisan's extractor on our raw data. The outputs from both extractors successfully pass all the standard statistical tests, which indicates that our postprocessing is effective in extracting out uniform randomness from a weak randomness source. All the test results are shown in Appendix B.

B. Autocorrelation

An alternative approach to verify randomness is evaluating the autocorrelation. The autocorrelations of the raw data are shown in Figs. 2(a) (between bits) and 2(b) (between samples). From Fig. 2(a), we can see that the autocorrelation is significant only within an eight-bit sample, but drop to the vicinity of below 1×10^{-3}. Also, the low values of the autocorrelation between samples [Fig. 2(b)] verify the assumption that the sequence of raw data is i.i.d. (see Sec. II A). We remark that due to the finite bandwidth of a practical detector and statistical fluctuations, the autocorrelation is around 1×10^{-3} but never drops to 0.

After postprocessing by either Travisan's extractor or the Toeplitz-hashing extractor, not only the correlation within eight bits (from a sample digitalized by an eight-bit ADC) is eliminated, but also the autocorrelation beyond eight bits drops to 1×10^{-5}. The autocorrelations of the postprocessing outputs are shown in Figs. 2(c) and 2(d), where the low residual values indicate the good randomness of our extracted results.

V. CONCLUDING REMARKS

We have modeled QRNG to evaluate the min-entropy of the quantum source, and discussed implementation of a popular extractor—Trevisan's extractor. We have also implemented a Toeplitz-hashing-based extractor. We have applied our postprocessing scheme to a recent QRNG implementation [22] and the min-entropy evaluation procedure on another implementation [12]. The random numbers obtained at the end of postprocessing passed through all the tests of DIEHARD, NIST, and TESTU01.

From our implementation of Trevisan's extractor, we find that the bottleneck for its extraction speed lies on the one-bit extractor part. Thus, in order to improve the implementation speed, one should investigate the one-bit extractor. In fact, the recent follow-up work by Maurer *et al.* shows that such improvement can be made [24]. Our prototypical implementation of Trevisan's extractor allows researchers to better understand the complexity and difficulty in the implementation of Trevisan's extractor, thus paving the way to future implementations.

ACKNOWLEDGMENTS

We thank C.-H. F. Fung and C. Rockoff for enlightening discussions. We also thank H. Zheng and N. Raghu for the preliminary work on the programming. Support from funding agencies NSERC, the CRC program, CIFAR, MaRS POP, and QuantumWorks is gratefully acknowledged. X.M. gratefully acknowledges the financial support from the National Basic Research Program of China Grants No. 2011CBA00300 and No. 2011CBA00301; National Natural Science Foundation of China Grants No. 61073174, No. 61033001, No. 61061130540, and No. 61003258; and the 1000 Youth Fellowship program in China.

APPENDIX A: TREVISAN'S EXTRACTOR IMPLEMENTATION DETAILS

The choice of block size not only determines the seed cost and security parameter of the random output, but also affects

[14]Toeplitz hashing can be implemented much faster with hardware implementation [20].

[15]www.stat.fsu.edu/pub/diehard/

[16]www.csrc.nist.gov/groups/ST/toolkit/rng/

TABLE I. A parameter set for Trevisan's extractor.

Extraction efficiency	RS GF(2^{m_e})	Design GF(2^{m_d})	Input	Output
$\rho = 1$	$m_e = 128$	$m_d = 8$	$n_i = 2^{15}$	$n_f = 2^{14}$
Security parameter	ECC codeword	Blocks	Seed	
$\varepsilon = \sqrt{2^{4-m_e} n_i n_f^2}$	$\bar{n} = 2^{2m_e}$	$b = 7$	$d = 4m_e^2 b$	

TABLE II. Real-time profile of the speed of combinatoric design. Parameters are selected to result in the highest generation rate. Number theoretical operations in GF_{2^m} dominate the speed performance of the ECC, and determine the speed of real-time performance and bit rate (per second).

$\log_2 n_f$	Experimental no. of GF_{2^m} operation	Theoretical no. of GF_{2^m} operation	Experimental no. of GF_{2^m} operation per n_f size	Real time (s)
10	65280	262144	63.75	41.1934
11	196352	786432	95.875	124.8
12	458496	2097152	111.9375	300.81
13	982784	5242880	119.96875	685.91
14	203130	12582912	123.984375	1603.8
15	4128512	29360128	125.99	3960.4
16	8322816	67108864	126.9960938	10911

TABLE III. Real-time profile of the speed of the error control code (ECC). Parameters are selected to result in the highest generation rate. Number-theoretical operations in GF_{2^m} dominate the speed performance of the ECC, and determine the speed of real-time performance and bit rate (per second).

n_f (power)	Experimental no. of GF_{2^m} operation	Theoretical no. of GF_{2^m} operation	Experiment no. of GF_{2^m} operation per n_f size	Real time (s)	Bit rate (s^{-1})
1024(10)	15360	16384	15	1.4488	706.8
2048(11)	63488	65536	31	5.9326	345.21121
4096(12)	258048	262144	63	23.5451	173.96401
8192(13)	1040384	1048576	127	95.72	173.96
16384(14)	4177920	4194304	255	380.19	43.1
32768(15)	16744482	16777216	511	1536.8	21.32

TABLE IV. DIEHARD. Data size is 240 Mbits. For the cases of multiple P values, a Kolmogorov-Smirnov (KS) test is used to obtain a final P value, which measures the uniformity of the multiple P values. The test is successful if all final P values satisfy $0.01 \leqslant P \leqslant 0.99$.

	Pseudo-RNG	Raw data	Trevisan's		Toeplitz hashing	
Statistical test	Result	Result	P value	Result	P value	Result
Birthday spacings (KS)	Success	*Failure*	0.822630	Success	0.340863	Success
Overlapping permutations	Success	*Failure*	0.679927	Success	0.403824	Success
Ranks of 31×31 matrices	Success	*Failure*	0.419095	Success	0.349441	Success
Ranks of 31×32 matrices	Success	*Failure*	0.715705	Success	0.816752	Success
Ranks of 6×8 matrices (KS)	Success	*Failure*	0.195485	Success	0.408573	Success
Bit stream test	Success	*Failure*	0.048260	Success	0.281680	Success
Monkey test OPSO	Success	*Failure*	0.027300	Success	0.892600	Success
Monkey test OQSO	Success	*Failure*	0.023200	Success	0.267200	Success
Monkey test DNA	*Failure*	*Failure*	0.038000	Success	0.736700	Success
Count 1's in stream of bytes	Success	*Failure*	0.380162	Success	0.639691	Success
Count 1's in specific bytes	*Failure*	*Failure*	0.020417	Success	0.373149	Success
Parking lot test (KS)	*Failure*	*Failure*	0.629013	Success	0.151689	Success
Minimum distance test (KS)	Success	*Failure*	0.019499	Success	0.688780	Success
Random spheres test (KS)	Success	*Failure*	0.488703	Success	0.939227	Success
Squeeze test	Success	*Failure*	0.238004	Success	0.155403	Success
Overlapping sums test (KS)	Success	*Failure*	0.022339	Success	0.909675	Success
Runs test (up) (KS)	*Failure*	*Failure*	0.403504	Success	0.181024	Success
Runs test (down) (KS)	Success	*Failure*	0.119132	Success	0.668512	Success
Craps test no. of wins	Success	*Failure*	0.757521	Success	0.826358	Success
Craps test throws per game	Success	*Failure*	0.179705	Success	0.862986	Success

TABLE V. NIST. Data size is 3.25 Gbits (500 sequences with each sequence around 6.5 Mbits). To pass the test, P value should be larger than the lowest significant level $\alpha = 0.01$, and the proportion of sequences satisfying $P > \alpha$ should be greater than 0.976. Where the test has multiple P values, the worst case is selected.

	Pseudo-RNG	Raw data	Toeplitz hashing		
Statistical test	Result	Result	P value	Proportion	Result
Frequency	Success	*Failure*	0.373625	0.9900	Success
Block frequency	Success	*Failure*	0.310049	0.9960	Success
Cumulative sums	Success	*Failure*	0.422638	0.9980	Success
Runs	Success	*Failure*	0.703417	0.9900	Success
Longest run	Success	*Failure*	0.013569	0.9880	Success
Rank	Success	*Failure*	0.411840	0.9940	Success
FFT	Success	*Failure*	0.987079	0.9860	Success
Nonoverlapping template	*Failure*	*Failure*	0.727851	0.9820	Success
Overlapping template	Success	*Failure*	0.110083	0.9780	Success
Universal	Success	*Failure*	0.962688	0.9880	Success
Approximate entropy	Success	*Failure*	0.674543	0.9920	Success
Random excursions	Success	*Failure*	0.409207	0.9900	Success
Random-excursions variant	Success	*Failure*	0.426358	0.9840	Success
Serial	Success	*Failure*	0.217570	0.9860	Success
Linear complexity	Success	*Failure*	0.657833	0.9940	Success

the complexity aspect of the performance. For demonstration purposes, we pick up a set of parameters for Trevisan's extractor, listed in Table I, which can be run sufficiently fast on a personal computer.

In this case, the random seed length is larger than the output length, and we can concatenate a hashing-based extractor to make the entropy loss minimum [18]. We pick up the output length of $n_f = 1$ Mb. On one hand, too large a n_f will slow down the extractor, much owing to the $O(n^2)$ complexity with respect to input length; on the other hand, too small a n_f will result in not only high seed cost but also a degradation of security (a larger security parameter ε).

Careful analysis of computational complexity is essential to understanding the tractability or intractability of our implementation given a reasonable computational power. The analysis of complexity of the combinatorial design in Table II demonstrates that the most economical parameter in terms of rate is at $n_f = 2^{14}$. A smaller parameter will render the design powerless due to associated high key cost, and a larger parameter results in unwieldy complexity growth.

As in Table III, the top generation rate of our extractor is 706.8 bits/s; the low speed of the extractor is a consequence of the lack of efficient implementation of finite field operations. Although slow in speed, the results from Trevisan's extractor do pass the statistical tests of DIEHARD. This increase in performance is at the cost of decrease in speed. The severe restriction on speed has limited the usage of Trevisan's extractor in real-time applications.

TABLE VI. TESTU01 (small crush). Given the constraint of the data size and computational power of crush and big crush of TESTU01, we only perform the small crush test here. Data size is 8 Gbits. The P value from a failing test converges to 0 or 1. Where the test has multiple P values, the worst case is selected.

	Pseudo-RNG	Raw data	Toeplitz hashing	
Statistical test	Result	Result	P value	Result
BirthdaySpacings	Success	*Failure*	0.5300	Success
Collision	Success	*Failure*	0.1500	Success
Gap Chi-square	Success	*Failure*	0.8900	Success
SimpPoker Chi-square	Success	*Failure*	0.3500	Success
CouponCollector Chi-square	Success	*Failure*	0.6700	Success
MaxOft Chi-square	Success	*Failure*	0.6900	Success
MaxOft Anderson-Darling	Success	*Failure*	0.9500	Success
WeightDistrib Chi-square	Success	*Failure*	0.5600	Success
MatrixRank Chi-square	Success	*Failure*	0.5100	Success
Hammingindep Chi-square	Success	*Failure*	0.1000	Success
RandomWalk1 H Chi-square	Success	*Failure*	0.9931	Success
RandomWalk1 M Chi-square	Success	*Failure*	0.8300	Success
RandomWalk1 J Chi-square	Success	*Failure*	0.9400	Success
RandomWalk1 R Chi-square	Success	*Failure*	0.7000	Success
RandomWalk1 C Chi-square	Success	*Failure*	0.6600	Success

Our implementation is done on a mere PC, but a mainframe computer can crunch number-theoretical operations much faster than a PC. Furthermore, as a future perspective, once we tackle the implementation on any graphical processing unit (GPU) platforms, the architecture of GPUs will allow us to exploit the intrinsic parallelism of the extractor much more efficiently via multithreading capability.

APPENDIX B: STATISTICAL TEST RESULTS

We employ three statistical tests—DIEHARD, NIST, and TESTU01 [34]—to evaluate the randomness of our extracted results from the Toeplitz-hashing extractor and Trevisan's extractor. The test results are shown in Tables IV–VI. We can see that the outputs from two extractors successfully pass all the standard statistical tests. Here, given the constraint of computational power for Trevisan's extractor, we skip the NIST and TESTU01 tests for its results. Without postprocessing, the raw data cannot pass any statistical tests, which is mainly due to the classical noises mixed in the raw data, and the fact that the measured quantum fluctuations follow Gaussian distribution instead of uniform distribution. This demonstrates the requirement of effective postprocessing in the QRNG.

For control purposes, we also perform the statistical tests on a pseudo-RNG generated from MATLAB2007. It generates uniformly random numbers from 0 to 255 (as emulation of eight-bit ADC output). The results are shown in Tables IV–VI. It cannot pass all tests.

[1] N. Meteopolis and S. Ulam, J. Am. Stat. Assoc. **44**, 335 (1949).

[2] N. Nisan and A. Wigderson, J. Comput. Syst. Sci. **49**, 149 (1994).

[3] C. H. Bennett and G. Brassard, in *Proceedings of the IEEE International Conference on Computers, Systems and Signal Processing* (IEEE, New York, 1984), pp. 175–179.

[4] B. Schneier and P. Sutherland, *Applied Cryptography: Protocols, Algorithms, and Source Code in C* (Wiley, New York, 1995).

[5] T. Jennewein, U. Achleitner, G. Weihs, H. Weinfurter, and A. Zeilinger, Rev. Sci. Instrum. **71**, 1675 (2000).

[6] A. Uchida *et al.*, Nat. Photonics **2**, 728 (2008).

[7] I. Reidler, Y. Aviad, M. Rosenbluh, and I. Kanter, Phys. Rev. Lett. **103**, 24102 (2009).

[8] R. Impagliazzo, L. A. Levin, and M. Luby, in *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, STOC '89* (ACM, New York, 1989), pp. 12–24.

[9] M. Epstein, L. Hars, R. Krasinski, M. Rosner, and H. Zheng, in *Cryptographic Hardware and Embedded Systems-CHES 2003*, Lecture Notes in Computer Science, Vol. 2779, edited by C. D. Walter, C. K. Koc, and C. Paar (Springer, Berlin Heidelberg, 2003) pp. 152–165.

[10] B. Qi, Y.-M. Chi, H.-K. Lo, and L. Qian, Opt. Lett. **35**, 312 (2010).

[11] M. A. Wayne and P. G. Kwiat, Opt. Express **18**, 9351 (2010).

[12] C. Gabriel, C. Wittmann, D. Sych, R. Dong, W. Mauerer, U. Andersen, C. Marquardt, and G. Leuchs, Nat. Photonics **4**, 711 (2010).

[13] B. Chor and O. Goldreich, in *26th Annual Symposium on Foundations of Computer Science, 1985* (IEEE, New York, 1985), pp. 429–442.

[14] D. Zuckerman, in *Proceedings of the 31st Annual Symposium on Foundations of Computer Science, 1990* (IEEE, New York, 1990), pp. 534–543.

[15] L. Trevisan, J. ACM **48**, 2001 (1999).

[16] R. Raz, O. Reingold, and S. Vadhan, in *Proceedings of the 31st Annual ACM Symposium on Theory of Computing,* 1999 (unpublished), pp. 149–158.

[17] A. De, C. Portmann, T. Vidick, and R. Renner, SIAM J. Comput. **41**, 915 (2012).

[18] R. Raz, O. Reingold, and S. Vadhan, J. Comput. Syst. Sci. **65**, 97 (2002).

[19] Y. Mansour, N. Nisan, and P. Tiwari, Theor. Comput. Sci. **107**, 235 (2002).

[20] H. Krawczyk, in *Advances in Cryptology - CRYPTO'94*, Lecture Notes in Computer Science Vol. 893 (Springer-Verlag, Berlin, 1994), pp. 129–139.

[21] C. H. Bennett, G. Brassard, C. Creäpeau, and U. M. Mauer, IEEE Trans. Inf. Theory **41**, 1905 (1995).

[22] F. Xu, B. Qi, X. Ma, H. Xu, H. Zheng, and H.-K. Lo, Opt. Express **20**, 12366 (2012).

[23] M. N. Wegman and J. L. Carter, J. Comput. Syst. Sci. **18**, 143 (1979).

[24] W. Mauerer, C. Portmann, and V. B. Scholz, arXiv:1212.0520.

[25] R. Canetti, Technical Report No. TR01-016, 2001 (unpublished).

[26] R. Canetti and H. Krawczyk, in *EUROCRYPT 2002*, Lecture Notes in Computer Science Vol. 2332 (Springer-Verlag, Berlin, 2002), pp. 337–351.

[27] M. Ben-Or, M. Horodecki, D. W. Leung, D. Mayers, and J. Oppenheim, in *Second Theory of Cryptography Conference TCC 2005*, Lecture Notes in Computer Science Vol. 3378 (Springer-Verlag, Berlin, 2005), pp. 386–406.

[28] R. Renner and R. König, in *Second Theory of Cryptography Conference TCC 2005*, Lecture Notes in Computer Science Vol. 3378 (Springer-Verlag, Berlin, 2005), pp. 407–425.

[29] X. Ma, Z. Zhang, and X. Tan, arXiv:1109.6147.

[30] T. Hartman and R. Raz, Rand. Struct. Alg. **23**, 235 (2003).

[31] X. Ma, C.-H. F. Fung, J.-C. Boileau, and H. Chau, Comput. Secur. **30**, 172 (2011).

[32] C.-H. F. Fung, X. Ma, and H. F. Chau, Phys. Rev. A **81**, 012318 (2010).

[33] T. Asai and T. Tsurumaru, IEICE Technical Report No. ISEC2010-121 2011 (in Japanese) (unpublished).

[34] P. L'Ecuyer and R. Simard, ACM Trans. Math. Softw. **33**, 22 (2007).