`randextract`: a Reference Library to Test and Validate Privacy Amplification Implementations

Iyán Méndez Veiga[1,2][a], Esther Hänggi[1][b]
[1]*School of Computer Science and Information Technology, Lucerne University of Applied Sciences and Arts, 6343 Rotkreuz, Switzerland*
[2]*Institute for Theoretical Physics, ETH Zurich, 8093 Zurich, Switzerland*
iyan.mendezveiga@hslu.ch

Keywords: Quantum Cryptography, Standardization, Privacy Amplification, Randomness Extractors, QKD, QRNG

Abstract: Quantum cryptographic protocols do not rely *only* on quantum-physical resources, they also require reliable classical communication and computation. In particular, the secrecy of any quantum key distribution protocol critically depends on the correct execution of the *privacy amplification* step. This is a classical post-processing procedure transforming a partially secret bit string, known to be somewhat correlated with an adversary, into a shorter bit string that is close to uniform and independent of the adversary's knowledge. It is typically implemented using randomness extractors. Standardization efforts in quantum cryptography have focused on the security of physical devices and quantum operations. Future efforts should also consider all algorithms used in classical post-processing, especially in privacy amplification, due to its critical role in ensuring the final security of the key. We present `randextract`, a reference library to test and validate privacy amplification implementations.

1 INTRODUCTION

Quantum cryptography is a multidisciplinary field taking advantage of quantum features, such as entanglement or the impossibility of cloning quantum states, to design cryptographic protocols. Two such protocols are Quantum Key Distribution (QKD) (Bennett and Brassard, 1984; Ekert, 1991) and Quantum Random Number Generators (QRNGs) (Stefanov et al., 2000). QKD allows two honest parties to establish a secret key over an insecure public quantum channel, and QRNGs can produce truly unpredictable randomness.

Besides the physical implementation using quantum technology, quantum cryptographic protocols rely on classical computation and communication. In particular, both QKD and QRNGs depend on classical data processing steps to achieve their goals. One of these steps, privacy amplification (Bennett et al., 1995), is crucial for security. Bugs or deviations from a correct implementation can introduce vulnerabilities and compromise the security guarantees of the protocols. In the case of QKD, this could result in the generation of insecure keys.

Current standardization efforts in quantum cryptography have focused on the security of physical devices and quantum operations (ISO/IEC JTC 1/SC 27, 2023; ETSI ISG-QKD, 2016). To ensure the security of the complete protocol, these efforts have to be extended to include all classical post-processing tasks. This includes the careful selection and standardization of algorithms, rigorous testing against publicly available test vectors, and validation through well-established certification programs. In contrast to classical cryptographic algorithms and devices (NIST CMVP, 2025; NIST CAVP, 2025), such processes do not yet exist for quantum cryptography.

1.1 Our Contribution

Our work provides a reference implementation of the functions used in privacy amplification allowing to check for correctness.

1. `randextract` is an open-source **Python library** that implements (modified) Toeplitz hashing and Trevisan's extractor, two types of functions commonly used in the privacy amplification step. It is distributed as an easy-to-use Python package. This allows the source code to remain close to the mathematical formulations, reduces the possi-

[a] https://orcid.org/0009-0004-8249-4661
[b] https://orcid.org/0000-0002-9760-8535

bility of implementation bugs due to peculiarities of the programming language and improves readability, allowing a large audience to read and audit the code. The library focuses on correctness, containing hundreds of unit and integration tests, and is not performance-optimized.

2. The library can be used directly in a quantum cryptographic protocol to **implement the privacy amplification step**.
3. The library can be used to test and **validate high-performance implementations**, whose source code may be unavailable or difficult to audit. We use `randextract` to test and validate three randomness extractors that have been used in recent QKD and QRNG implementations. With these real world examples, we highlight the importance of the classical post-processing phase for security and correctness.
4. We provide a function to generate **test vectors** for privacy amplification implementations in order to facilitate external audits. We use the same format as in the NIST Cryptographic Algorithm Validation Program. These test vectors can be used by the community in a validation process included into future standardization efforts of quantum cryptographic devices.

1.2 Outline

The paper is organized as follows: Sec. 2 introduces the background of quantum key distribution, quantum random number generation and privacy amplification. In particular, Sec. 2.4 introduces the theory of two families of quantum-proof extractors: (modified) Toeplitz hashing and Trevisan's construction. Sec. 3, 4, and 5 present our main contributions. In Sec. 3, we describe our Python package, `randextract`, in detail. Sec. 4 demonstrates how the library can be used to test and validate third-party implementations. Notably, using our library, we identify and correct a bug in a modified Toeplitz hashing implementation used in a high-speed QKD experiment. We also uncover discrepancies between the mathematical specification and the actual implementation of a high-performance Trevisan's construction implementation used in QKD and QRNG experiments.

Sec. 5 introduces a set of test vectors for Toeplitz hashing, proposed as a step towards future standardization efforts. Finally, Sec. 6 presents our conclusions and outlines directions for future work.

2 THEORETICAL BACKGROUND

2.1 Notation & Definitions

The capital letters X, Y and Z denote *classical random variables*, which take values x, y and z. Calligraphic fonts $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ denote the *alphabet* of the corresponding random variables. The *probability* that the random variable Z takes the value z is $P_Z(Z=z)$, sometimes abbreviated to $P_Z(z)$ or $P(z)$ when it is clear from the context which random variable it refers to. For two random variables X and Y, the *joint probability* is defined as $P_{XY}(x,y) = P(X = x \wedge Y = y)$, and the *conditional probability* of X given Y as $P_{X|Y}(x,y) = P_{X|Y=y}(x) = P_{XY}(x,y)/P_Y(y)$.

The *guessing probability*, defined as the probability of correctly guessing the value of a random variable X, is

$$p_{\text{guess}}(X) = \max_{x\in\mathcal{X}} P_X(X=x)\,,$$

which motivates the definition of the *min-entropy*

$$H_{\min}(X) = -\log_2 p_{\text{guess}}(X)\,.$$

The *uniform* distribution over a random variable X is denoted by U_X, i.e., each $x \in \mathcal{X}$ is equally likely, and the guessing probability is $1/|\mathcal{X}|$.

The *statistical distance* between two probability distributions P_X and P_Y, defined over the same alphabet $\mathcal{X}$, is given by

$$d(X,Y) = \frac{1}{2}\sum_{x\in\mathcal{X}} |P_X(x=x) - P_Y(y=x)|\,.$$

Operationally, the statistical distance represents the maximum advantage one can gain in correctly identifying whether a single sample was drawn from P_X or P_Y. A probability distribution of a random variable X is said to be $\varepsilon-$*close to uniform* if its distance with respect to the uniform distribution is bounded by ε, i.e.,

$$d(X,U) = \frac{1}{2}\sum_{x\in\mathcal{X}} \left| P(x) - \frac{1}{|\mathcal{X}|} \right| \leq \varepsilon\,. \tag{1}$$

Greek letters, such as ρ and σ, are used to denote quantum states. Given a Hilbert space $\mathcal{H}$, a *quantum state* ρ can be represented by a positive semi-definite Hermitian operator of unit trace acting on $\mathcal{H}$. Any quantum state can be expressed in terms of an orthonormal basis $\{|i\rangle\}_i$ of $\mathcal{H}$

$$\rho = \sum_{i,j} \rho_{ij}\, |i\rangle\langle j|\,,$$

where ρ_{ij} are complex numbers, $\rho_{ij} = \rho^*_{ji}$, and $\mathrm{Tr}[\rho] = \sum_i \rho_{ii} = 1$.

Letters from the beginning of the alphabet, such as A, B or E are used to denote the subsystems that form a *composite quantum system*. For example, ρ_{ABE} denotes a quantum state acting on the Hilbert space $\mathcal{H} = \mathcal{H}_A \otimes \mathcal{H}_B \otimes \mathcal{H}_E$.

A (classical) random variable X can be written as a *diagonal* quantum state

$$\rho_X = \sum_{x \in \mathcal{X}} P(x) \left|x\right\rangle\!\left\langle x\right| ,$$

and the uniform distribution can be represented as the *maximally mixed state*

$$\sigma_U = \frac{1}{\dim(\mathcal{H})} \sum_{i=1}^{\dim(\mathcal{H})} \left|i\right\rangle\!\left\langle i\right| .$$

Of special interest in quantum cryptography are the so-called *classical-quantum (cq) states*, which allow to represent a quantum system correlated with a classical random variable. Given a probability distribution P_X and a set of quantum states $\{\rho_E^x\}_x$, a cq-state is defined as

$$\rho_{XE} = \sum_{x \in \mathcal{X}} \left(P_X(x) \left|x\right\rangle\!\left\langle x\right| \otimes \rho_E^x\right) . \tag{2}$$

This formalism is particularly relevant in security analyses, where the classical variable models the honest party's output (e.g., a raw key), and the quantum system E represents the adversary's *side information*.

Quantum measurements are represented by a set of positive operators $\{E_i\}_i$ satisfying $\sum_i E_i = \mathbb{1}$. These sets are called *Positive Operator-Valued Measure* (POVM). Given a state ρ and a measurement $\{E_i\}_i$, the probability of measuring the value i is given by $\mathrm{Tr}[E_i \rho]$.

The *trace distance* generalizes the notion of statistical distance to the quantum setting. For two quantum states ρ and σ, it is defined as

$$d(\rho,\sigma) = \frac{1}{2} \left\|\rho - \sigma\right\|_{\mathrm{tr}} = \frac{1}{2} \mathrm{Tr} \sqrt{(\rho-\sigma)^\dagger (\rho-\sigma)} .$$

Analogously to the statistical distance, the trace distance provides an operational interpretation as a measure of distinguishability between quantum states. If a system is prepared in either ρ or σ with equal probability, the maximum probability with which the state can be correctly identified; e.g., by doing a (single) quantum measurement or feeding the state into another quantum protocol, is $\frac{1}{2} + \frac{1}{2} d(\rho,\sigma)$.

The trace distance of a classical random variable X from uniform from the point of view of an adversary with quantum side information E is given by

$$d(\rho_{XE}, \sigma_U \otimes \rho_E) = \frac{1}{2} \left\|\rho_{XE} - \sigma_U \otimes \rho_E\right\|_{\mathrm{tr}} . \tag{3}$$

Finally, both the guessing probability and the min-entropy can be generalized to the setting of cq-states. The guessing probability of a random variable X conditioned on a quantum system E is

$$p_{\mathrm{guess}}(X|E) = \max_{\{E_x\}} \sum_{x \in \mathcal{X}} P_X(x) \,\mathrm{Tr}[\rho_E^x E_x] ,$$

where the maximization is over all the POVMs, and the conditional min-entropy

$$H_{\min}(X|E) = -\log_2 p_{\mathrm{guess}}(X|E) .$$

2.2 Quantum Cryptography

2.2.1 Quantum Key Distribution

An *ideal* key distribution protocol is one that produces a bit string that is identical for the honest parties and that looks uniform from the adversary's point of view. *Real* protocols cannot usually achieve this perfectly and, instead, produce an output that is ε-close to the ideal one, i.e., the trace distance between the output of the real and the ideal protocol, as in Eq. (3), is bounded by ε. The trace distance guarantees that the real protocol is indistinguishable from the ideal protocol, except with probability ε. This notion of security as the distinguishability between an ideal and a real protocol guarantees that the protocol is *composable* (Pfitzmann and Waidner, 2001; Backes et al., 2003; Canetti, 2001; Maurer, 2002), i.e., it remains secure even if it is used with other protocols to form a larger cryptographic system.

Key distribution is traditionally implemented using public-key cryptography, e.g., RSA (Rivest et al., 1983), Diffie-Hellman (Diffie and Hellman, 1976) or ML-KEM (Avanzi et al., 2020; National Institute of Standards and Technology, 2024). These protocols are secure if the adversary is limited in computing power, and given some assumptions on the computational hardness of certain mathematical problems.

Quantum Key Distribution (QKD) is a quantum cryptographic protocol that enables two honest parties to establish a shared secret key over an insecure quantum channel[1]. QKD is fundamentally different from public-key cryptography because the established keys are information-theoretically secure (Maurer, 1999), i.e., their security does not depend on a limitation in computing power of the adversary.

Any quantum key distribution protocol consists of two distinct phases:

[1] In addition to the public quantum channel, the honest parties need to be able to communicate classically over an authentic channel and have access to local randomness. Their labs need to be secured and isolated, and depending on the protocol, there might be additional conditions on the quantum devices.

1. a quantum phase, in which quantum states are prepared, transmitted over a public quantum channel, and measured;
2. and a classical phase, in which the classical data that corresponds to the preparation settings and measurement results is post-processed to derive the final shared secret keys.

The classical post-processing involves several subprotocols or steps, which might differ slightly from one particular protocol to another. A common way to implement this phase in a QKD protocol is the following:

1. The starting point after the quantum phase are two *raw keys*, bit strings held by the honest parties that are neither equal nor completely secret.
2. A sample from the raw keys is compared and then discarded to perform the *testing* step, also known as *parameter estimation*. The honest parties reveal some of the bits over the classical channel to estimate the losses and number of errors, which allows to bound the amount of information an adversary might have obtained during the quantum phase[2]. Testing aborts the protocol if this estimation is beyond a certain threshold, and continues otherwise. A real implementation correctly recognizes when an adversary has too much information except with probability ε_{pe}.
3. In case the honest parties decide to continue the protocol, the next step is called *information reconciliation* and corrects the errors of the (remaining) raw keys to obtain the *corrected keys*. This step ensure the correctness of the protocol, and it is often implemented using error-correcting codes (see e.g., (Peterson and Weldon, 1972)). After this step, the two honest parties hold identical keys, which are partially secret, except with probability ε_{ir}.
4. Lastly, the corrected keys are transformed into shorter but completely secret bit strings in the *privacy amplification* step. These are the final *secret keys*. At the end of this step, the honest parties still hold equal keys but these are now also secret from the adversary except with probability ε_{pa}.

A *security proof* of a QKD protocol (Renner, 2005) is a theoretical statement, with well-defined conditions and a security claim in terms of a *security parameter* ε bounding the probability that the output of the protocol differs from an ideal key distribution protocol, i.e., guarantees that the honest parties end up with an equal, perfectly secure key except with probability ε. If the subprotocols are defined in a composable way, then the security parameter can be bounded in terms of the parameters of these subprotocols using the union bound (Boole, 1847), i.e., $\varepsilon \leq \varepsilon_{\text{pe}} + \varepsilon_{\text{ir}} + \varepsilon_{\text{pa}}$. This security parameter ε can ideally be made arbitrarily small at the cost of a lower key rate.

[2] In reality, errors and losses might be due to the noisy quantum channel. In cryptography, it is common to assume the worst-case scenario. In this case, any error is attributed to an adversary eavesdropping and manipulating the transmitted quantum states.

2.2.2 Quantum Random Number Generator

A Quantum Random Number Generator (QRNG) is a type of hardware random number generator that uses quantum systems to generate randomness. Most hardware random number generators use classical physical processes, and since these processes are deterministic, these protocols need to assume that the initial state of the system is only partially known by an adversary. QRNGs, on the other hand, take advantage of the inherent unpredictability of quantum phenomena. Even if a quantum state is perfectly described, the outcome of a measurement is, in general, not deterministic (Born, 1955).

QRNG protocols are similar to QKD protocols but with just one party involved. In the quantum phase, the same party prepares and measures quantum states, and the classical post-processing does not contain an information reconciliation step. The security notion for QRNG protocols is the same as in QKD, i.e., close-to-uniform randomness even from the point of view of an adversary who might hold a quantum state initially correlated with the state of the QRNG system. The privacy amplification step can be used in exactly the same way as in a QKD protocol. It removes bias arising from the quantum phase and side-information held by an adversary.

2.3 Privacy Amplification with Seeded Randomness Extractors

As introduced in Sec. 2.2.1, Privacy Amplification (PA) is a classical post-processing procedure whose goal is to *compress* partially secret (classical) bit strings into bit strings that are uniform and independent of any adversary's (quantum) knowledge. In the context of QKD, the adversary's knowledge of the corrected keys can originate from eavesdropping or tampering with the quantum states transmitted over the public quantum channel, as well as from observing the authenticated classical communication, such as the error correction syndrome exchanged during the information reconciliation step.

The correct composable definition of ideal keys

Figure 1: Diagram of a seeded randomness extractor that takes an n-bit string and a uniform d-bit uniform string as inputs, and outputs an ε-close to uniform m-bit string.

or ideal randomness after privacy amplification is an upper bound on Eq. (3), i.e.,

$$d(\rho_{XE}, \sigma_U \otimes \rho_E) \leq \varepsilon_{\text{pa}}\,,$$

which guarantees that the final keys from QKD or the output from a QRNG look uniform from an adversary's point of view, except with a small probability ε_{pa}.

Seeded Randomness Extractors Privacy amplification can be implemented using seeded randomness extractors (see e.g., (Shaltiel, 2004)). A *seeded randomness extractor* is a function that takes as input

1. a weak random source X, i.e., a bit string of length n, and
2. a uniform seed Y, i.e., a bit string of length d,

and outputs a bit string of length m (see Figure 1), i.e.,

$$\text{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m \qquad (4)$$

A seeded randomness extractor can be used to transform a bit string with high min-entropy into an almost-uniform bit string. In order to correctly implement the privacy amplification step using these extractors we require two additional conditions:

1. The output of the extractor is independent of the seed. This is because the random seed is revealed to the adversary during the execution of the protocol.
2. The output is close to uniform for an adversary with quantum side information.

Extractors satisfying the first condition are called *strong*, and those satisfying the second condition *quantum-proof*[3].

A quantum-proof (k,ε)-strong seeded randomness extractor (König and Renner, 2011) is a function defined as in Eq. (4) which, for a uniform independent seed Y and input X with conditional min-entropy $H_{\min}(X|E) \geq k$, outputs a string $\text{Ext}(X,Y)$ satisfying

$$d(\rho_{\text{Ext}(X,Y)YE}, \sigma_U \otimes \rho_Y \otimes \rho_E) \leq \varepsilon_{\text{pa}}\,, \qquad (5)$$

[3]If the output of the extractor is close to uniform for an adversary with classical side information, the extractor is said to be *classical-proof*. All quantum-proof extractors are classical-proof, but the converse is not true.

i.e., the output is ε_{pa}-close to the ideal output which is uniform, independent of the seed and uncorrelated to the quantum side information.

In the context of QKD and QRNG protocols, only quantum-proof strong randomness extractors can to be used to realize the privacy amplification step. Therefore, these are the extractors implemented in `randextract`.

2.4 Construction of Randomness Extractors

Families of two-universal hash functions are quantum-proof strong extractors. Alternative constructions are possible such as the Trevisan's construction.

A family of functions $\mathcal{F} = \{f_y\}_y$, such that $f_y : \mathcal{X} \to \mathcal{Z}$, is *two-universal* if

$$\Pr_{f_y \in \mathcal{F}}[f_y(x) = f_y(x')] \leq \frac{1}{|\mathcal{Z}|}\,,$$

for any distinct $x, x' \in \mathcal{X}$, and f_y picked uniformly at random (Carter and Wegman, 1979). This means that the number of collisions, when the function is picked uniformly at random, is bounded by the size of the output alphabet. There exist two-universal functions from $\{0,1\}^n$ to $\{0,1\}^m$ for any positive integers $0 \leq m \leq n$ (Wegman and Carter, 1981).

A seeded randomness extractor, as defined in Eq. (4), is realized from a family of two-universal hash functions $\mathcal{F}$ by using the seed $y \in \mathcal{Y}$ to choose one particular function $f_y \in \mathcal{F}$ and hashing the input $x \in \mathcal{X}$ with this chosen function, i.e.,

$$\text{Ext}(x,y) := f_y(x)\,. \qquad (6)$$

The quantum leftover hash lemma (Renner and König, 2005; Renner, 2005; Tomamichel et al., 2011) states that a family of two-universal functions, as defined in Eq. (6), is a quantum-proof $(k,\varepsilon_{\text{pa}})$-strong extractor for output length m less or equal than

$$\left\lfloor k + 2 - 2\log\frac{1}{\varepsilon} \right\rfloor .$$

2.4.1 Toeplitz Hashing

The set of all binary $m \times n$ Toeplitz matrices defines a family of two-universal hash functions, where each matrix corresponds to a specific function in the family. A matrix T is said to be a Toeplitz matrix if its elements satisfy

$$T_{i,j} = T_{i-1,j-1}\,. \qquad (7)$$

Toeplitz matrices are, therefore, fully characterized by their first row and column.

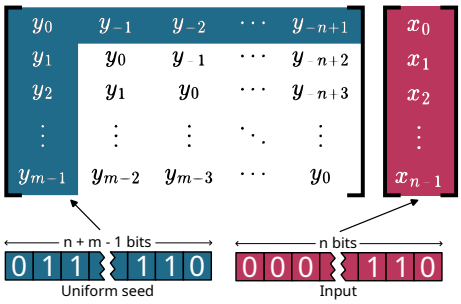

Figure 2: Toeplitz hashing is defined as the matrix-vector multiplication of a Toeplitz matrix generated from a uniform seed y of length $n+m-1$ and input x of length n from a weak random source.

One common method to select a specific matrix from this family using the seed y is to define the matrix entries based on y as follows:

$$T_{i,j}(y) := y_{i-j}\,. \tag{8}$$

Standard Toeplitz Hashing Using Eq. (8) to construct a matrix from the seed, the standard Toeplitz hashing (see e.g., (Krawczyk, 1995)) is defined as

$$\mathrm{Ext}_{\mathrm{Toeplitz}}(x,y) := T(y)\cdot x\,. \tag{9}$$

Figure 2 shows this construction graphically.

Efficient implementation General matrix-vector multiplication has a computational complexity $O(n^2)$. However, the matrix-vector multiplication of Toeplitz hashing can be implemented more efficiently with a computational complexity $O(n\log n)$ using the Fast Fourier Transform (FFT). First, the $m\times n$ Toeplitz matrix is transformed into a square $(m+n-1)\times(m+n-1)$ circulant matrix by adding $n-1$ additional rows and $m-1$ columns. Then, this matrix can be diagonalized using the Fourier matrix F_q, whose matrix elements are determined by $F_{j,k}=\frac{1}{\sqrt{q}}e^{2\pi ijk/q}$ with $q=m+n-1$. The circulant matrix can, therefore, be written in terms of the seed y as

$$\hat{T}_q(y) = F_q^{-1}\mathrm{diag}(F_q y)F_q\,.$$

And the complete Toeplitz extractor as

$$\mathrm{Ext}_{\mathrm{Toeplitz}}(x,y) = \mathrm{FFT}^{-1}\Big(\mathrm{FFT}(y)\odot\mathrm{FFT}(\hat{x})\Big)\Big|_{0\ldots m-1},$$

where $\hat{x}$ is the input x padded with $m-1$ zeros, $\odot$ denotes element-wise multiplication of the two vectors, and $|_{0\ldots m-1}$ means that the output of the inverse fast Fourier transform is truncated to the first m bits (see App. C from (Hayashi and Tsurumaru, 2016)).

Modified Toeplitz Hashing Standard Toeplitz hashing requires a seed of length $m+n-1$ bits. However, it is possible to reduce the seed length to $n-1$ bits by using a different family of two-universal hash functions (Hayashi and Tsurumaru, 2016). For the same input and output lengths, the so-called *modified Toeplitz hashing* uses as its hashing matrix the concatenation of a smaller Toeplitz matrix with the $m\times m$ identity matrix, i.e.,

$$\mathrm{Ext}_{\mathrm{Mod.\ Toeplitz}}(x,y) := (T'(y)\,\|\,\mathbb{1}_m)x\,, \tag{10}$$

where $T'(y)$ is a Toeplitz matrix of dimension $m\times(n-m)$. The matrix-vector multiplication remains efficiently computable using the same FFT-based approach described above.

2.4.2 Trevisan's Construction

Two-universal hashing is not the only way to define strong quantum-proof seeded extractors. Trevisan (Trevisan, 2001) developed a method to construct arbitrary extractors from one-bit extractors and weak designs. Later, it was proven that this construction generates a quantum-proof strong extractor (De et al., 2012) from a strong one-bit extractor.

One-Bit Extractors One-bit extractors are simply extractors defined as in Eq. (4) with $m=1$. Any one-bit (k,ε)-strong extractor is a quantum-proof $(k-\log\varepsilon, 3\sqrt{\varepsilon})$-strong extractor. If the one-bit extractor is already classical-proof, then it can be shown that it is also a quantum-proof $(k,(1+\sqrt{2})\sqrt{\varepsilon})$-strong extractor (König and Terhal, 2008).

A combinatorial design (Nisan and Wigderson, 1994) is a family of subsets $W=[S_0,S_1,\ldots,S_{m-1}]$, with $S_i\subseteq[d]$, typically constructed so that the intersections between the subsets satisfy certain constraints. These designs are widely used in theoretical computer science due to their ability to balance overlap and independence.

Weak Design A weak (m,t,r,d)-design is a combinatorial design, i.e., a family of sets $W=[S_0,S_1,\ldots,S_{m-1}]\in[d]$, where all sets are of size t, satisfying

$$\sum_{j=0}^{i-1} 2^{|S_i\cap S_j|} \le rm \tag{11}$$

for all i (Raz et al., 2002).

Trevisan's extractor Given a quantum-proof strong one-bit extractor $\mathrm{Ext}_1(x,y)$ and a weak design

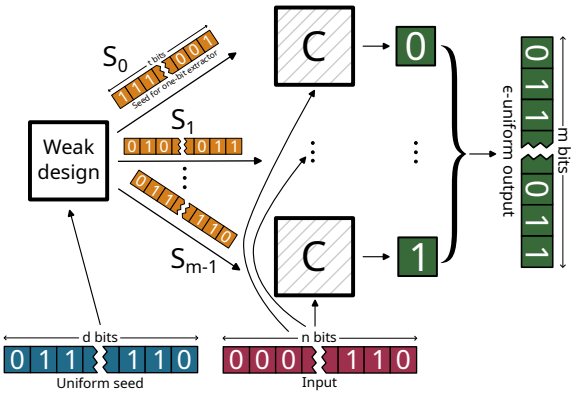

Figure 3: Trevisan's extractor output (12) is the concatenation of the m bits obtained from calling the one-bit extractor m times on the same input but with a different seed each time. The m seeds $y_{S_0}, \ldots, y_{S_{m-1}}$ are constructed from the input seed y using the sets from the weak design as indices.

$W = [S_0, \ldots, S_{m-1}]$, Trevisan's extractor is realized as

$$\mathrm{Ext}_{\mathrm{Trevisan}}(x, y) := \mathrm{Ext}_1(x, y_{S_0}) \ldots \mathrm{Ext}_1(x, y_{S_{m-1}}) . \tag{12}$$

Figure 3 shows this construction graphically.

Computational complexity The computational complexity of Trevisan's construction depends on the specific choice of the weak design and the one-bit extractor. In practice, Trevisan's extractor is often significantly slower than Toeplitz hashing. However, it offers an important advantage: certain constructions require only a seed of polylogarithmic length in the input size. For instance, the construction using a weak design over a finite field and a polynomial-time one-bit extractor requires a seed of length $O(\log^2 n)$ (Mauerer et al., 2012). In contrast, Toeplitz and modified Toeplitz hashing require seeds of length $O(n)$. Trevisan's extractor may therefore be preferable in scenarios where good randomness is a scarce resource.

3 RANDEXTRACT

`randextract` is our open-source Python package implementing some of the most used privacy amplification algorithms. It is available at the Python Package Index (PyPI) (Mendez Veiga and Hänggi, 2025). We recommend installing it in a virtual Python environment, for example[4], by running the following commands

```
python -m venv --upgrade-deps env-randextract
source env-randextract/bin/activate
pip install randextract
```

Other ways of installing the package are described in the latest available online documentation[5].

3.1 Overview & Goals

The goal of `randextract` is to provide a readable and easy-to-audit library implementing most of the PA algorithms used in quantum cryptographic protocols. The source code stays close to the mathematical formulation to avoid deviations and facilitate the audit of its correctness. Only when it can be done without impacting this primary goal, secondary goals such as performance are taken into consideration. The package also provides classes and helping functions to aid validating third-party implementations. The package is intended to be self-contained, with comprehensive documentation that introduces the theory of randomness extractors, explains their relevance to quantum cryptographic protocols, includes several toy examples suitable for manual computation, and provides real-world validation cases.

3.2 Structure & Design

The source code repository follows a standard Python structure.

- `src/randextract`: source code of the library.
- `tests`: unit and integration tests.
- `docs/source`: source code of the online documentation[5].
- `examples`: scripts validating real world privacy amplification implementations.
- `resources`: additional resources such as plots, datasets used in testing and the scripts to generate them, Jupyter notebooks, test vectors, etc.

The package is modular by design allowing to be easily extended in the future with additional extractors. A common API is enforced using abstract classes. For example, the abstract class `RandomnessExtractor` defined in the module `randomness_extractor.py` requires that the implementation classes, such as the `ToeplitzHashing` class, implement the properties `input_length`, `seed_length` and `output_length`, and the method `extract()`.

[4] In Windows, the activation of the virtual environment should be done running `env-randextract\Scripts\Activate.ps1` instead.

[5] https://randextract.crypto-lab.ch

3.3 Dependencies

`randextract` uses the Python Standard Library (Python Software Foundation, 2024) and the well-known numerical libraries NumPy (Harris et al., 2020) and Scipy (Virtanen et al., 2020). In addition, the Galois package (Hostetter, 2020) is used to compactly define arrays and polynomials over finite fields.

3.4 Usage

`randextract` can be used directly to implement the PA step of quantum cryptographic protocols. It can also be used to validate other implementations. In both cases the workflow is the same:

1. Choose a particular class of seeded randomness extractors.
2. Instantiate one particular extractor with the desired input and output lengths, and the required parameters. When using `randextract` to implement the PA step, the optimal parameters can be calculated with the library. When validating other implementations, these have to match the implementation being tested.
3. Extract the randomness from the weak source, or validate the implementation.

Implementing PA The choice of the extractor class in Step 1 depends on the requirements of the quantum cryptographic protocol. In Step 2, the optimal input and output lengths can be computed using `randextract`, based on the model of the weak randomness source and the desired security parameter ε_{pa} from Eq. (5). This is handled by the `calculate_length()` helper method available in any `RandomnessExtractor` implementation. Finally, the output is computed in Step 4 by calling the `extract()` method on the instantiated extractor.

Validating other implementations To validate an external implementation, Steps 1 and 2 must follow the same extractor class and parameter choices. In Step 3, rather than manually extracting outputs and comparing them across different seeds and inputs, the custom extractor can be passed directly to the `Validator` class. External implementations are registered using the `add_implementation()` method, and validation is performed via the `validate()` method. Any failing test cases are recorded, and detailed insights into discrepancies can be obtained using the `analyze_failed_test()` method.

3.5 Code Examples

3.5.1 Toeplitz Hashing

Implementing PA with Toeplitz hashing The following code snippet illustrates how to create an extractor based on standard Toeplitz hashing, as defined in Eq. (9). First, the relevant classes are imported:

```
from galois import GF2

import randextract
from randextract import (
    RandomnessExtractor,
    ToeplitzHashing
)
```

Next, the optimal output length is computed for inputs of 8 Mib originating from a weak random source with initial min-entropy $\frac{1}{n}H_{\min}(X|E) \geq \frac{1}{2}$, and for a security parameter of $\varepsilon_{\text{pa}} = 10^{-6}$.

```
Mib = 2**20

out_len = ToeplitzHashing.calculate_length(
    extractor_type="quantum",
    input_length=8*MiB,
    relative_source_entropy=0.5,
    error_bound=1e-6,
)
```

A Toeplitz hashing extractor is then instantiated for the corresponding input and output lengths:

```
ext = RandomnessExtractor.create(
    extractor_type="toeplitz",
    input_length=8*MiB,
    output_length=out_len
)
```

Finally, the hashing is applied to (pseudo)random input and seed bit strings:

```
ext_int = GF2.Random(MiB)
ext_seed = GF2.Random(ext.seed_length)
ext_out = ext.extract(ext_int, ext_seed)
```

Validating a third-party implementation of modified Toeplitz hashing The following code snippet demonstrates how to validate a third-party implementation. First, two additional classes are imported:

```
from randextract import (
    ModifiedToeplitzHashing,
    Validator
)
```

The reference extractor is instantiated using the `ModifiedToeplitzHashing` class, which implements Eq. (10), with the same input and output lengths as the implementation under test. In this case, the extractor takes inputs of 1 Mib and compresses them

by 50%. The resulting object is functionally equivalent to one created via `RandomnessExtractor.create()`, as shown in the previous example.

```
ref_ext = ModifiedToeplitzHashing(
    input_length=2**20,
    output_length=2**19
)
```

An auxiliary function is defined to convert binary arrays into string representations:

```
def gf2_to_str(gf2_arr):
    arr = np.array(gf2_arr)
    arr_str = (arr + ord("0")).tobytes()
    return arr_str.decode()
```

A `Validator` instance is then initialized with the reference extractor, and the third-party implementation is registered. The validator is configured to interact with the implementation via standard input/output, using the previously defined conversion function as a parser:

```
val = Validator(ext)
val.add_implementation(
    label="Rust-stdio-fft",
    input_method="stdio",
    command="./modified_toeplitz $SEED$ $INPUT$",
    format_dict={
        "$SEED$": gf2_to_str,
        "$INPUT$": gf2_to_str
    },
)
```

Validation is performed using 10^4 randomly generated input samples:

```
val.validate(
    mode="random",
    sample_size=10**4
)
```

3.5.2 Trevisan's extractor

To instantiate a randomness extractor based on Trevisan's construction, additional parameters must be specified. In addition to the input and output lengths, common to all seeded extractors, one must select concrete implementations for the one-bit extractor and the weak design. These implementations may impose constraints on the parameters. For instance, the weak design provided by the class `FiniteFieldPolynomialDesign` requires the size of the subsets to be a prime number (or a prime power). Such constraints are automatically handled by the `calculate_length()` method.

The following example shows how to instantiate a Trevisan extractor configured to take 1 Mib of input and output 1 Kib, using a finite field weak design and a polynomial-based one-bit extraction. No additional imports are required:

```
ext = RandomnessExtractor.create(
    extractor_type="trevisan",
    weak_design_type="finite_field",
    one_bit_extractor_type="polynomial",
    one_bit_extractor_seed_length=1024,
    input_length=2**20,
    output_length=2**10,
)
```

Once the extractor object has been instantiated, the extraction process is performed identically across all implementations:

```
ext_input = GF2.Random(ext.input_length)
ext_seed  = GF2.Random(ext.seed_length)
ext_out = ext.extract(ext_input, ext_seed)
```

Additional examples and the complete API documentation for `randextract` are available online[6].

4 VALIDATING PRIVACY AMPLIFICATION IMPLEMENTATIONS

`randextract` is well-suited for small proof-of-concept implementations or protocols where privacy amplification can be performed offline. In contrast, production environments requiring real-time privacy amplification typically demand performance-optimized and highly efficient implementations. In the last years, due to huge improvements on the quantum-physical part of quantum cryptography protocols (Grünenfelder, 2022), the classical post-processing has become a bottleneck (Yuan et al., 2018). The secret key rate of QKD protocols and the throughput of QRNGs is limited by how fast the classical post-processing can be done. This has motivated the development of high-performance implementations of both information reconciliation and PA algorithms, using hardware accelerators such as GPUs (Bosshard et al., 2021) or FPGAs (Li et al., 2019). These implementations are harder to read and audit, and deviations from the mathematical definitions are harder to spot.

In this section we show how we used `randextract` to test, validate and fix such high-performance PA implementations.

4.1 GPU modified Toeplitz hashing

Modern GPUs can accelerate a wide range of computational tasks, including the calculation of the FFT. A CUDA and Vulkan-based implementation of the modified Toeplitz hashing algorithm (Bosshard

[6] https://randextract.crypto-lab.ch/api.html

et al., 2021) was employed in a QKD experiment (Grünenfelder et al., 2023) involving ultrafast single-photon detectors, where constraints on block lengths and throughput made the use of FPGAs impractical. To the best of our knowledge, this remains the fastest known implementation of modified Toeplitz hashing.

We tested the GPU-based implementation using our own extractor alongside the `Validator` class. Instead of relying on files for input and output, we integrated our package with the GPU implementation using ZeroMQ queues, which are its preferred communication interface. During validation, we immediately observed discrepancies in approximately half of the tests involving randomized inputs, specifically those where the last bit was set to 1. This allowed us to quickly identify the root cause of the issue: the GPU implementation was ignoring the final bit of the input vector.

Our library helped us in identifying and fixing the issue. The GPU implementation is available online[7] and our validation is provided as an example in the repository[8].

4.2 Rust Toeplitz hashing

Rust is a modern programming language that is designed to achieve both memory safety and high performance (Klabnik and Nichols, 2019). Memory safety is a valuable feature for all software, but it is especially critical for software used in cryptographic protocols since many serious security bugs are caused by memory related issues (Cybersecurity and Infrastructure Security Agency, 2023). To explore this further, a Rust implementation of Toeplitz hashing was developed at the Lucerne University of Applied Sciences and Arts as part of a semester student project[9]. The primary goal was to evaluate the performance of a CPU-based Toeplitz hashing implementation within a memory-safe environment.

We used the `Validator` class to exhaustively test all possible input–seed pairs for small input lengths by interacting directly with the Rust process. For larger input vectors, we conducted randomized tests using file-based input and output. The library was employed throughout the development of the Rust implementation for continuous testing, to support identifying bugs and handling edge cases.

[7] https://github.com/nicoboss/PrivacyAmplification

[8] https://github.com/cryptohslu/randextract/blob/main/examples/validation_gpu_modified_toeplitz_zeromq.py

[9] Source code is available on request.

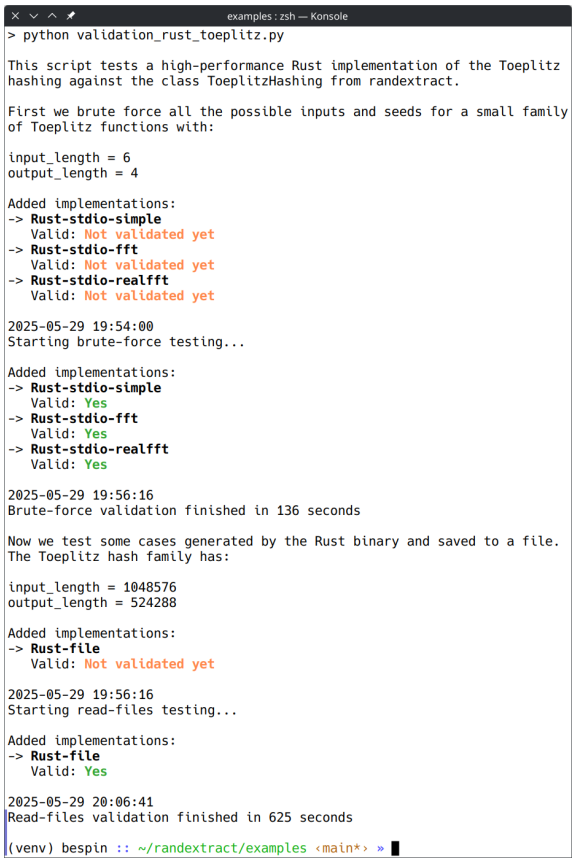

Figure 4: Screenshot showing the output of the script `validation_rust_toeplitz.py`, available in the `examples` directory in our repository, testing the Rust implementation. First, all possible inputs and seeds were tested with very small Toeplitz matrices. Then, random samples with larger inputs and outputs were validated.

4.3 C++ Trevisan's construction

Validating Trevisan's extractors is more complex than validating extractors based on two-universal hashing. This increased complexity arises from the greater number of configurable components in Trevisan's construction, such as the choice of one-bit extractor and weak design.

A C++ high-performance implementation of Trevisan's construction providing two different weak designs and three one-bit extractors was developed in (Mauerer et al., 2012). This library was used in QRNG (Kavuri et al., 2024) and QKD (Nadlinger et al., 2022) experiments.

When validating this implementation, we identified three issues affecting the basic weak design implementation based on finite fields. Two of these are caused by an over-optimization of the multiplication and addition operations, which are implemented using left-shift and bitwise OR operations, respectively.

While such optimizations are correct for powers of two, they are generally invalid when working with finite fields of arbitrary order, such as those used in this implementation. As a result, the computed subset overlap violates the bound specified in Eq. (11), indicating that the resulting family of subsets does not form a valid weak design. Consequently, the Trevisan's construction, as defined in Eq. (12), using these subsets is not a strong quantum-proof extractor. The third issue concerns a deviation in how polynomial evaluation is performed. In particular, the coefficients are interpreted in reverse compared to the formulation in App. C.1 of (Mauerer et al., 2012). We believe this deviation does not affect the correctness of the weak design, but it does result in outputs that differ from our implementation.

A full description of the issues and solutions are contained directly in the library repository[10].

5 TEST VECTORS FOR PRIVACY AMPLIFICATION

Classical cryptographic functions are heavily standardized. There exist standards and procedures to get certified (see e.g., (Alagic et al., 2025)). The National Institute of Standards and Technology (NIST) operates the Cryptographic Module Validation Program (CMVP), which promotes the use of validated cryptographic modules. The validation proceeds in two steps:

1. First, the underlying cryptographic algorithms and their components are tested and validated through the Cryptographic Algorithm Validation Program (CAVP) (NIST CAVP, 2025). This validation is performed using the Automated Cryptographic Validation Test System in a black-box manner. The implementation receives a list of inputs in a request file (`.req`). The implementation then obtains the outputs based on those inputs and generates a response file (`.rsp`), which contains both the provided inputs from the received request and the outputs.
2. Then, the full cryptographic module, including software or hardware integration, is tested under the CMVP (NIST CMVP, 2025). This step involves functional testing, documentation review, and an evaluation of the module's conformance to requirements defined in standards such as FIPS 140-3 (National Institute of Standards and Technology, 2019). The testing is performed by accredited third-party laboratories, and successful validation results in an official NIST certificate, authorizing the module for use in regulated and security-sensitive environments.

Other countries, such as Spain (Centro Criptológico Nacional, 2020) in Europe and South Korea (National Intelligence Service, Republic of Korea, 2015) in Asia, have similar validation processes for cryptographic modules, typically based on the ISO/IEC 19790 standard (International Organization for Standardization and International Electrotechnical Commission, 2025).

Currently, no equivalent validation program exists for the algorithms used in quantum cryptographic devices. Most efforts have focused on the physical layer, particularly on mitigating side-channel attacks. Looking ahead, it is crucial to establish validation frameworks for the classical post-processing steps of QKD, especially for the steps that are crucial for the final security such as parameter estimation or privacy amplification, since these are the steps that guarantee the secrecy of the final key. Standardization of QKD protocols should define a set of approved PA algorithms, which would then be subject to testing in a CAVP-like program using randomly generated test vectors. These vectors should comprehensively reflect the range of inputs that the device is expected to handle in practice, taking into account both its operational capabilities and the constraints dictated by the protocol's security proof. After the successful completion of these tests, the post-processing modules can be validated under a CMVP-style certification program.

The `Validator` class in `randextract` can generate tests vectors following the same format as the request (`.req`) and response (`.rsp`) files used in CAVP.

5.1 Example: Test Vectors for Toeplitz Hashing

As an example, we provide below a response file for a small modified Toeplitz hashing extractor. Additional larger examples are available online[11]. Request and response files for any randomness extractor implemented in `randextract` can be generated using the `generate_test_vector()` method from the `Validator` class.

```
# CAVS
# ModifiedToeplitzHashing
# Input Length : 128
# Compression ratio: 1/2
# Generated on Tue May 20 15:12:03 2025

[EXTRACT]
```

[10] https://github.com/wolfgangmauerer/libtrevisan/pull/2

[11] https://github.com/cryptohslu/randextract/resources/test_vectors

```
COUNT = 0
INPUT = e3fc097a6dcc77fc781a7ed3533528c8
SEED = 05f47ea39db462da99e3e29b06721ae6
OUTPUT = ab264a34f8ebc27c

COUNT = 1
INPUT = ff3d1bfe1f4c15730dc6ec1c36c7c4e8
SEED = 3eeaf730861d37e9d751d29fd6ad0ece
OUTPUT = 6411c793f97badae

COUNT = 2
INPUT = b7fa3c803d20709f25603bb1b3072917
SEED = 63296df538784e26c446211c058eb9a4
OUTPUT = 3bcfb106e23573e2

COUNT = 3
INPUT = 42c2ddcaef33a3e7998104c76605a588
SEED = 05b7c4012ffc8b5a17cdc544f3e7e2fd
OUTPUT = 48f041d38296ffcc

COUNT = 4
INPUT = a14c3632e4fbffff0e10b10ba4ccdc5d
SEED = 7733fabb766a34b3883762e240db6f20
OUTPUT = 16b0ed99752aa43a

COUNT = 5
INPUT = 23473c65a2c5ab8dbe073f8e419ccee7
SEED = 0c50697d5a102b6ef9016e809fb6a515
OUTPUT = f0b5b4d1f7cb519f

COUNT = 6
INPUT = 5b2719a61b8f72e208587b4ad0ec8ac0
SEED = 0ee322c8bfa4a7e901b3e0bcb0f8bad3
OUTPUT = b1731fb59a4bdb98

COUNT = 7
INPUT = 82c6f364c42caa101fb70e562585fc86
SEED = 29aa29456ea804ca102737d1d150e221
OUTPUT = d35034bccd12b0c4
```

6 CONCLUSIONS & FUTURE WORK

The correctness of classical post-processing, and in particular the privacy amplification step, is essential for the security of real quantum cryptographic protocols. Only through careful and rigorous validation can the keys obtained in QKD protocols be secure in practice. Deviations from correct privacy amplification procedures may enable practical attacks, allowing adversaries, even those without quantum capabilities, to partially or fully recover the secret keys.

Our main contribution is an open-source Python library that implements quantum-proof strong extractors used in QKD and QRNG protocols. The library is designed for readability and auditability, with implementations closely following the mathematical definitions. We used this library to test, validate, and fix issues in external high-performance implementations.

Future work can add new classes of functions to `randextract` such as quantum-proof two-source (seedless) extractors. The modular design of the library will facilitate this task. Additionally, `randextract` can provide an open platform to test and validate quantum cryptographic protocols in ongoing post-processing standardization efforts at ETSI, ITU-T, etc.

ACKNOWLEDGMENTS

This work was supported by the Swiss National Science Foundation Practice-to-Science Grant No 199084.

REFERENCES

Alagic, G., Bros, M., Ciadoux, P., Cooper, D., Dang, Q., Dang, T., Kelsey, J., Lichtinger, J., Liu, Y.-K., Miller, C., Moody, D., Peralta, R., Perlner, R., Robinson, A., Silberg, H., Smith-Tone, D., and Waller, N. (2025). Status report on the fourth round of the nist post-quantum cryptography standardization process. Technical Report NIST IR 8545, National Institute of Standards and Technology. Accessed May 30, 2025.

Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., and Stehlé, D. (2020). CRYSTALS-Kyber: Algorithm specifications and supporting documentation. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions. Third-round submission to the NIST Post-Quantum Cryptography Standardization Process.

Backes, M., Pfitzmann, B., and Waidner, M. (2003). A composable cryptographic library with nested operations. In *CCS'03: Proceedings of the ACM Conference on Computer and Communications Security*, pages 220–230.

Bennett, C., Brassard, G., Crepeau, C., and Maurer, U. (1995). Generalized privacy amplification. *IEEE Transactions on Information Theory*, 41(6):1915–1923.

Bennett, C. H. and Brassard, G. (1984). Quantum cryptography: Public key distribution and coin tossing. In *Proceedings of IEEE International Conference on Computers, Systems, and Signal Processing*, page 175, India.

Boole, G. (1847). *The Mathematical Analysis of Logic: Being an Essay Towards a Calculus of Deductive Reasoning*. Macmillan, Cambridge, UK. Reprinted by Philosophical Library.

Born, M. (1955). Statistical interpretation of quantum mechanics. *Science*, 122(3172):675–679.

Bosshard, N., Christen, R., Hänggi, E., and Hofstetter, J. (2021). Fast privacy amplification on gpus. Poster presentation at the 24th Annual Conference on Quantum Information Processing (QIP 2021), Online.

Canetti, R. (2001). Universally composable security: a new paradigm for cryptographic protocols. In *FOCS '01: Proceedings of the Symposium on Foundations of Computer Science*, pages 136–145.

Carter, J. and Wegman, M. N. (1979). Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154.

Centro Criptológico Nacional (2020). Esquema de evaluación y certificación de la seguridad de las tecnologías de información. https://oc.ccn.cni.es/documentos/normativa-y-legislacion/51-po-005-certificacion-de-productos-en/file. Accessed May 30, 2025.

Cybersecurity and Infrastructure Security Agency (2023). The urgent need for memory safety in software products. https://www.cisa.gov/news-events/news/urgent-need-memory-safety-software-products. Accessed May 30, 2025.

De, A., Portmann, C., Vidick, T., and Renner, R. (2012). Trevisan's extractor in the presence of quantum side information. *SIAM Journal on Computing*, 41(4):915–940.

Diffie, W. and Hellman, M. E. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654.

Ekert, A. K. (1991). Quantum cryptography based on Bell's theorem. *Physical Review Letters*, 67(6):661–663.

ETSI ISG-QKD (2016). ETSI GS QKD 011 V1.1.1 — Quantum Key Distribution (QKD); Component characterization: characterizing optical components for QKD systems. Group Specification GS QKD 011 V1.1.1, European Telecommunications Standards Institute, Sophia Antipolis, France.

Grünenfelder, F. (2022). *Performance, Security and Network Integration of Simplified BB84 Quantum Key Distribution*. PhD thesis, Université de Genève. Available at https://archive-ouverte.unige.ch/unige:164897.

Grünenfelder, F., Boaron, A., Resta, G. V., Perrenoud, M., Rusca, D., Barreiro, C., Houlmann, R., Sax, R., Stasi, L., El-Khoury, S., Hänggi, E., Bosshard, N., Bussières, F., and Zbinden, H. (2023). Fast single-photon detectors and real-time key distillation enable high secret-key-rate quantum key distribution systems. *Nature Photonics*, 17(5):422–426.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with numpy. *Nature*, 585(7825):357–362.

Hayashi, M. and Tsurumaru, T. (2016). More efficient privacy amplification with less random seeds via dual universal hash function. *IEEE Transactions on Information Theory*, 62(4):2213–2232.

Hostetter, M. (2020). Galois: A performant NumPy extension for Galois fields. https://github.com/mhostetter/galois.

International Organization for Standardization and International Electrotechnical Commission (2025). ISO/IEC 19790:2025: Information technology — Security techniques — Security requirements for cryptographic modules. https://www.iso.org/standard/52906.html. Accessed May 30, 2025.

ISO/IEC JTC 1/SC 27 (2023). ISO/IEC 23837-2:2023 — Information security — Security requirements, test and evaluation methods for quantum key distribution — Part 2: Evaluation and testing methods. International Standard ISO/IEC 23837-2:2023, International Organization for Standardization and International Electrotechnical Commission, Geneva. Published September 2023.

Kavuri, G. A., Palfree, J., Reddy, D. V., Zhang, Y., Bienfang, J. C., Mazurek, M. D., Alhejji, M. A., Siddiqui, A. U., Cavanagh, J. M., Dalal, A., Abellán, C., Amaya, W., Mitchell, M. W., Stange, K. E., Beale, P. D., Brandão, L. T. A. N., Booth, H., Peralta, R., Nam, S. W., Mirin, R. P., Stevens, M. J., Knill, E., and Shalm, L. K. (2024). Traceable random numbers from a nonlocal quantum advantage.

Klabnik, S. and Nichols, C. (2019). *The Rust Programming Language*. No Starch Press, 2 edition.

König, R. and Renner, R. (2011). Sampling of min-entropy relative to quantum knowledge. *IEEE Transactions on Information Theory*, 57(7):4760–4787.

König, R. T. and Terhal, B. M. (2008). The bounded-storage model in the presence of a quantum adversary. *IEEE Transactions on Information Theory*, 54(2):749–762.

Krawczyk, H. (1995). New hash functions for message authentication. In Guillou, L. C. and Quisquater, J.-J., editors, *Advances in Cryptology — EUROCRYPT '95*, pages 301–310, Berlin, Heidelberg. Springer Berlin Heidelberg.

Li, Q., Yan, B.-Z., Mao, H.-K., Xue, X.-F., Han, Q., and Guo, H. (2019). High-speed and adaptive fpga-based privacy amplification in quantum key distribution. *IEEE Access*, 7:21482–21490.

Mauerer, W., Portmann, C., and Scholz, V. B. (2012). A modular framework for randomness extraction based on trevisan's construction.

Maurer, U. (1999). Information-theoretic cryptography. In Wiener, M., editor, *Advances in Cryptology — CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 47–64. Springer-Verlag.

Maurer, U. (2002). Indistinguishability of random systems. In *EUROCRYPT '02: Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*, pages 110–132.

Mendez Veiga, I. and Hänggi, E. (2025). randextract: A python reference implementation for testing and validating privacy amplification algorithms. https://pypi.org/project/randextract/. Version 0.2.0.

Nadlinger, D. P., Drmota, P., Nichol, B. C., Araneda, G., Main, D., Srinivas, R., Lucas, D. M., Ballance, C. J., Ivanov, K., Tan, E. Y.-Z., Sekatski, P., Urbanke, R. L., Renner, R., Sangouard, N., and Bancal, J.-D. (2022). Experimental quantum key distribution certified by bell's theorem. *Nature*, 607(7920):682–686.

National Institute of Standards and Technology (2019). Security requirements for cryptographic modules. Technical Report FIPS PUB 140-3, U.S. Department of Commerce. Accessed May 30, 2025.

National Institute of Standards and Technology (2024). Module-Lattice-Based Key-Encapsulation Mechanism Standard. Federal Information Processing Standards Publication (FIPS) NIST FIPS 203, Department of Commerce, Washington, D.C.

National Intelligence Service, Republic of Korea (2015). Korean cryptographic module validation program (kcmvp). https://eng.nis.go.kr/EAF/1_7_2_1.do. Accessed May 30, 2025.

Nisan, N. and Wigderson, A. (1994). Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167.

NIST CAVP (2025). Cryptographic Algorithm Validation Program (CAVP). Program Overview CAVP, National Institute of Standards and Technology. Programme website, accessed 30 May 2025.

NIST CMVP (2025). Cryptographic Module Validation Program (CMVP). Program Overview CMVP, National Institute of Standards and Technology. Programme website, accessed 30 May 2025.

Peterson, W. W. and Weldon, Jr., E. J. (1972). *Error-Correcting Codes*. MIT Press, Cambridge, MA, second edition.

Pfitzmann, B. and Waidner, M. (2001). A model for asynchronous reactive systems and its application to secure message transmission. In *SP '01: Proceedings of the 2001 IEEE Symposium on Security and Privacy*, page 184.

Python Software Foundation (2024). Python Language Reference, version 3.x. https://docs.python.org/3/. Accessed May 30, 2025.

Raz, R., Reingold, O., and Vadhan, S. (2002). Extracting all the randomness and reducing the error in trevisan's extractors. *Journal of Computer and System Sciences*, 65(1):97–128.

Renner, R. (2005). *Security of Quantum Key Distribution*. PhD thesis, ETH Zurich. Available at https://arxiv.org/abs/quant-ph/0512258.

Renner, R. and König, R. (2005). Universally composable privacy amplification against quantum adversaries. In *TCC'05: Proceedings of the Theory of Cryptography Conference*, pages 407–425.

Rivest, R. L., Shamir, A., and Adleman, L. M. (1983). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 26(1):96–99.

Shaltiel, R. (2004). *Recent developments in explicit constructions of extractors*, page 189–228. World scientific.

Stefanov, A., Gisin, N., Guinnard, O., Guinnard, L., and Zbinden, H. (2000). Optical quantum random number generator. *Journal of Modern Optics*, 47(4):595–598.

Tomamichel, M., Schaffner, C., Smith, A., and Renner, R. (2011). Leftover hashing against quantum side information. *IEEE Trans. Inform. Theory*, 57(8):5524–5535.

Trevisan, L. (2001). Extractors and pseudorandom generators. *J. ACM*, 48(4):860–879.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, I., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., Vijaykumar, A., Bardelli, A. P., Rothberg, A., Hilboll, A., Kloeckner, A., Scopatz, A., Lee, A., Rokem, A., Woods, C. N., Fulton, C., Masson, C., Häggström, C., Fitzgerald, C., Nicholson, D. A., Hagen, D. R., Pasechnik, D. V., Olivetti, E., Martin, E., Wieser, E., Silva, F., Lenders, F., Wilhelm, F., Young, G., Price, G. A., Ingold, G.-L., Allen, G. E., Lee, G. R., Audren, H., Probst, I., Dietrich, J. P., Silterra, J., Webber, J. T., Slavič, J., Nothman, J., Buchner, J., Kulick, J., Schönberger, J. L., de Miranda Cardoso, J. V., Reimer, J., Harrington, J., Rodríguez, J. L. C., Nunez-Iglesias, J., Kuczynski, J., Tritz, K., Thoma, M., Newville, M., Kümmerer, M., Bolingbroke, M., Tartre, M., Pak, M., Smith, N. J., Nowaczyk, N., Shebanov, N., Pavlyk, O., Brodtkorb, P. A., Lee, P., McGibbon, R. T., Feldbauer, R., Lewis, S., Tygier, S., Sievert, S., Vigna, S., Peterson, S., More, S., Pudlik, T., Oshima, T., Pingel, T. J., Robitaille, T. P., Spura, T., Jones, T. R., Cera, T., Leslie, T., Zito, T., Krauss, T., Upadhyay, U., Halchenko, Y. O., and Vázquez-Baeza, Y. (2020). Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature Methods*, 17(3):261–272.

Wegman, M. N. and Carter, J. (1981). New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279.

Yuan, Z., Plews, A., Takahashi, R., Doi, K., Tam, W., Sharpe, A. W., Dixon, A. R., Lavelle, E., Dynes, J. F., Murakami, A., Kujiraoka, M., Lucamarini, M., Tanizawa, Y., Sato, H., and Shields, A. J. (2018). 10-mb/s quantum key distribution. *Journal of Lightwave Technology*, 36(16):3427–3433.