Cryptomite:
A versatile and user-friendly library of randomness extractors

Cameron Foreman[1,2], Richie Yeung[3,4], Alec Edgington[5], and Florian J. Curchod[5]

[1]Quantinuum, Partnership House, Carlisle Place, London SW1P 1BX, United Kingdom
[2]Department of Computer Science, University College London, London, United Kingdom
[3]Quantinuum, 17 Beaumont Street, Oxford OX1 2NA, United Kingdom
[4]Department of Computer Science, University of Oxford, Oxford, United Kingdom
[5]Quantinuum, Terrington House, 13–15 Hills Road, Cambridge CB2 1NL, United Kingdom

We present `Cryptomite`, a Python library of randomness extractor implementations. The library offers a range of two-source, seeded and deterministic randomness extractors, together with parameter calculation modules, making it easy to use and suitable for a variety of applications. We also present theoretical results, including new extractor constructions and improvements to existing extractor parameters. The extractor implementations are efficient in practice and tolerate input sizes of up to $2^{40} > 10^{12}$ bits. Contrary to alternatives using the fast Fourier transform, we implement convolutions efficiently using the number-theoretic transform to avoid rounding errors, making them well suited to cryptography. The algorithms and parameter calculation are described in detail, including illustrative code examples and performance benchmarking.

Contents

Cameron Foreman: cameron.foreman@quantinuum.com
Richie Yeung: richie.yeung@quantinuum.com
Florian J. Curchod: florian.curchod@quantinuum.com

1 Introduction

Informally, a randomness extractor is a function that outputs *(near-)perfect* randomness (in the sense that it is almost uniformly distributed) when processing an input that is 'somewhat' random (in the sense that it may be far from uniformly distributed). Randomness extractors play an essential role in many cryptographic applications, for example as exposure-resilient functions, to perform privacy amplification in quantum key distribution or to distil cryptographic randomness from the raw output of an entropy source. Beyond cryptography, randomness extractors are a useful tool for many tasks, including the de-randomisation of algorithms or constructing list-decodable error-correcting codes. For an introduction to randomness extraction, see [1].

Despite their usefulness, a major difficulty is to select the appropriate randomness extractor and its parameters for the task at hand – and more generally, to design, optimise and implement the algorithm without significant expertise and time investment. To address this challenge, we have developed `Cryptomite`, a software library that offers state-of-the-art randomness extractors that are easy to use (see Section 5: 'Cryptomite: Examples with Code') and efficient (see Section 2.3: 'Performance'). Our implementations are also numerically precise, in the sense that the extractor algorithms do not use any floating point arithmetic that can lead to rounding errors. This is accomplished using the number-theoretic transform (NTT) [2], instead of the fast Fourier transform (FFT) [3], whenever implementing convolutions efficiently.[1] This is important since uncontrolled errors can lead to implementation vulnerabilities or failures, making our implementations more reliable than alternatives.

The library contains extractors from existing work and our own constructions, along with several improvements to parameters and security proofs. With the library, we provide explanations, theorems and parameter derivation modules to assist the user and ensure that they make a good extractor choice for their application. `Cryptomite` is made available at https://github.com/CQCL/cryptomite under a public licence https://github.com/CQCL/cryptomite/blob/main/LICENSE for non-commercial use, and installable through `pip`, with command `pip install cryptomite`. In [4], we additionally show how to use the extractors in `Cryptomite` to post-process the output of several (quantum) random

[1] The NTT can be seen as a generalisation of the FFT to finite fields, allowing to perform fast convolutions on integer sequences. The FFT directly performs fast convolutions using floating point numbers and thus precision limited by the amount of memory dedicated to it.

number generators and study their effect using intense statistical testing.

In the following, we give an overview of different types of randomness extractor (Section 1.1) and define our contributions (Section 1.2). Then, we present the `Cryptomite` library (Section 2) at a high level, describing each extractor and benchmarking the library's performance, as well as providing a simple, visual guide to help a user in selecting the appropriate extractor. After that, we present the library in full detail (Section 4), where we give useful theorems and lemmas related to randomness extraction and explain how to calculate all relevant parameters. We finish with code examples (Section 5) on how to use `Cryptomite` extractors in experimental quantum key distribution and randomness generation, as well as the improvements that are achieved by using extractors of `Cryptomite`.

1.1 Randomness extraction at a glance

Randomness extractors transform strings of weakly random numbers (which we call the *source* or *input*), in the sense that their distribution has some min-entropy (the most conservative measure of a random variable's unpredictability, see Definition 1), into (near-)perfect randomness, in the sense that the output's distribution is (near-)indistinguishable from uniform (see Definitions 3 and 4), as quantified by an extractor error ϵ. Some extractors require an additional independent string of (near-)perfect random numbers for this transformation, which we call the *seed*[2]. Additionally, we call the seed a *weak seed* if it is not (near-)perfect and has only a min-entropy guarantee.

In the manuscript, we use the notation n_1, n_2 to denote the lengths (in bits) of the weakly random extractor input and the (weak) seed respectively. We use k_1, k_2 (respectively) to denote their min-entropy, m the extractor's output length, ϵ the extractor error and $O(\cdot)$ the asymptotic quantities related to the extractor algorithm, for example its computation time. The set-up and variables are illustrated in Figure 1.

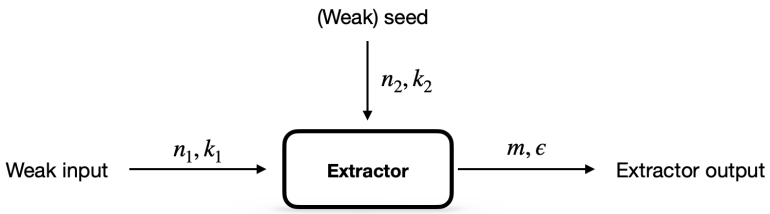

Figure 1: An extractor processes a weak input of n_1 bits with min-entropy k_1 in order to produce an output of m bits that is ϵ-perfectly random. Seeded and two-source extractors additionally require a second random bit string called the (weak) seed, of n_2 bits and min-entropy k_2 (with $k_2 = n_2$ for seeded extractors).

At a high level, randomness extractors can be described by three classes:

- **Deterministic Extractors** are algorithms that process the weak input alone, without additional resources. Such extractors are able to extract from a subset of weakly random sources only, but require certain properties of the input distribution beyond just a min-entropy promise – for example that every bit is generated in an independent and identically distributed (IID) manner. A list of works detailing the different types of input which can be deterministically extracted from can be found in [5] (Subsection 'Some related work on randomness extraction').
- **Seeded Extractors** require an additional string of (near-)perfect randomness, called a seed, which is generated independently of the extractor weak input. By leveraging this additional

[2]Remark that, for useful seeded extractors, the seed is either short or the extractor is strong (which means that its output is (almost) independent of the seed and therefore the seed is not *consumed* in the process, see Definition 8 for the formal statement).

independent randomness provided by the seed, they can extract from weak random inputs characterised by min-entropy only.

- **Multi-Source Extractors**, sometimes called blenders, are a generalisation of seeded extractors. They require at least one additional independent weak source of randomness, rather than a single (near-)perfect and independent one. In this work, the only multi-source extractors we consider are **two-source**, i.e. where the user has one addition weak input string only, which we call the weak seed.

When using randomness extraction algorithms in real-world protocols, the following features are crucial:

Implementability and efficiency: Some randomness extractors only have non-constructive proofs and are therefore theoretical objects that can only be used to derive fundamental results. Moreover, although a randomness extractor may have an explicit construction, this does not always mean that the algorithm can be implemented with a computation time suitable for the application at hand. For example, taking directly the building blocks in [6] would lead to a computation time of $O(n_1^4)$ (for $n_1 \propto n_2$), which significantly limits the maximum input lengths that the implementation can handle. For most applications, a computation time of $O(n_1^2)$ is a requirement, whilst many even require quasi-linear computation time $O(n_1 \log(n_1))$ – such as quantum key distribution. For an excellent discussion and concrete example of the need for quasi-linear computation time, see Appendix E, Section C of [7] and our examples with code in Section 5.

Additional resources: Seeded and two-source extractors require a seed, which is either near-perfect (seeded extractors) or weak (two-source extractor). In addition to the weakly random input to extract from, this obliges the user to have access to, or be able to generate, an independent (weak) seed, which can be difficult to justify (and often not even discussed). Because of this, it is important to minimise the resources required by the extractor, for example, by extracting with a (near-)perfect seed that is only short [7, 8] or minimising the entropy requirement on the weak seed [9, 10].

Security options: Certain randomness extractor constructions have been shown to fail against a quantum adversary who can store information about extractor inputs in quantum states [11]. Because of this, it is important to have the option to choose extractors that are *quantum-proof*, for example, in protocols such as quantum key distribution. Similarly, in certain protocols the adversary is able to perform actions that may correlate the input and the (weak) seed, therefore breaking the independence condition. This is the case for example in randomness amplification or could happen in quantum key distribution. Consequently, techniques have been developed that allow to weaken the independence condition [12–14].

Entropy loss: Entropy loss is the difference between the min-entropy of the extractor input and the length of the extractor output. Some randomness extractors are able to extract with entropy loss that is only logarithmic in the inverse extractor error, which is the fundamental minimum [15] against both quantum and classical adversaries.

In `Cryptomite`, we implemented several randomness extractors that have a variety of the important features presented above. Our extractors are efficient, with $O(n_1 \log n_1)$ or $O(n_1^2 \log^2 n_1)$ computation time, and have minimal or near-minimal entropy loss. They can extract from various initial resources and come with many options; such as whether to make them quantum-proof or secure when the input and (weak) seed are not fully independent. Additionally, all of our extractors are *strong* (meaning the output is statistically independent of one input), information-theoretically secure (secure against computationally unbounded adversaries) and *universally composable* [16] (enabling secure integration into broader cryptographic protocols).

1.2 Our contributions

The contributions of our work are:

1. We provide an efficient code implementation of a two-source Dodis et al. extractor, based on [17] and our previous work [18]. It is implemented in quasi-linear computation time $O(n_1 \log n_1)$. To our knowledge, there exist no alternative implementation.

2. We present a new strong seeded extractor construction, which we call Circulant, due to the input being used to construct a circulant matrix. Beyond its simplicity, it is directly quantum-proof with the same error (and equivalently, the same output length)[3], making it the best choice against quantum adversaries. It achieves the same entropy loss, quasi-linear computation time and error as Toeplitz-based ones [20], whilst requiring a seed that is only a single bit longer than the weak input length – making it an excellent choice for tasks such as quantum key distribution. This new extractor construction also has no alternative implementation.

3. We implement the Toeplitz extractor [20] efficiently (in $O(n_1 \log(n_1))$) and a Trevisan extractor, based on [8] and [21], which tolerates the shortest seed lengths for sufficiently large input lengths but is less efficient ($O(mn_1 \log^2 n_1)$ computation time, where m is the output size of the extractor – making it impractical in many cases)[4] We also provide an implementation of the deterministic Von Neumann extractor in $O(n_1)$.

4. Contrary to all alternative implementations, we implement the Circulant, Dodis et al. and Toeplitz extractors using the number-theoretic transform (NTT) implemented in C++, which gives a throughput of approximately 1Mbit/s for input sizes up to 5×10^8 on a standard personal laptop (see Section 2.3). Using the NTT instead of the fast Fourier transform (FFT) avoids numerical imprecision (due to floating point arithmetic) which may be unsuitable for some applications, e.g. in cryptography, especially with large input lengths. Contrary to alternatives, our software is able to process input lengths of up to $2^{40} > 10^{12}$ bits, which should be sufficient even for device-independent protocols [18, 23–28]. When using input lengths below $2^{29} \approx 5 \cdot 10^8$, the code uses a further optimised (smaller finite field for the) NTT to increase the throughput.

5. We collate (and sometimes improve, see for example Corollary 1) results from existing works into a single place, providing techniques to get the most (near-)perfect randomness in different adversarial models and experimental settings. For example, all our constructions can, as an option, extract under a weaker independence requirement on the weak seed (in the Markov model [13]) and we provide the associated parameters. Moreover, in the software we include `from_params` utility functions which calculate the input and output lengths for a number of settings and a `suggest_extractor` function, which assists a user on deciding which extractor to use for their application, based on the flow chart in Figure 2.

To the best of our knowledge, existing works only achieve a small subset of the features mentioned above. There are no alternative implementations of the Dodis et al. extractor or the Circulant extractor, which are new to this work. Reference [29] is the only work we found that offers a library containing several software implementations of randomness extractors, including the Von Neumann and Toeplitz extractors, as well as one termed 'universal hashing' and providing a link to the implementation of the Trevisan extractor [30]. However, it does not implement the Toeplitz extractor efficiently by exploiting the convolution theorem to obtain quasi-linear computation time, i.e., it does not use the FFT or NTT.

[3] The new construction is now a two-universal family of hash functions, allowing to apply well-known security proof techniques, as such, the results in [19] prove it is quantum-proof with the same output length in the seeded case.

[4] We note that the seed length is $O(\log(n_1))$, but this does not always imply a short seed in practice. For instance, in our implementation with near-minimal entropy loss, the seed length exceeds the input length when $n_1 < 10^6$ and $m \approx n_1/2$. Furthermore, it is often useful to assume that the output length m is proportional to the input length n_1, i.e., $m \propto n_1$. In this case, the computation time for the Trevisan extractor becomes $O(n_1^2 \log^2 n_1)$, making it inefficient. However, when m is small, Trevisan's extractor can have a reasonable computation time, as seen in [22], where $m = 512$.

Other software implementations of individual extractors exist. There are several publicly available implementations of the Toeplitz extractor, for example, in [31, 32] (in Python) and [33] (in C++ and Verilog for FPGA), but none utilise the convolution theorem for quasi-linear computation time – making them inefficient at larger input sizes. Implementations of the Trevisan extractor can be found in [34], related to [21] and [30], both written in C++. The implementation of [34] is particularly flexible, providing various instantiations of the extractor that can be useful to experts. In [35], the authors discuss and implement a modified Toeplitz extractor, based on [7], which is an extractor with parameters similar to Circulant. This implementation uses the FFT, enabling input lengths up to 10^7. However, by not using the NTT as we do, potential rounding errors may occur. A more detailed comparison is provided in Section 4.3.

2 Overview of the extractor library

In this section, we present an overview of the `Cryptomite` extractors, an informal guide for selecting an appropriate extractor for an application, and performance benchmarking. `Cryptomite` is implemented in Python for usability and ease of installation, with performance-critical parts implemented in C++ called from Python.

2.1 Extractors of `Cryptomite`

The `Cryptomite` library contains:

- The (new) Circulant strong seeded extractor. It requires a prime seed length of $n_2 = n_1 + 1$ and outputs $m = \lfloor k_1 - 2\log_2(\frac{1}{\epsilon})\rfloor$ bits against both classical and quantum adversaries (called classical- and quantum-proof respectively). We offer different two-source extensions of the extractor, one of which has output length $m = \lfloor k_1 + k_2 - n_2 - 2\log_2(\frac{1}{\epsilon})\rfloor$ against an adversary with quantum side information that is product on the two sources (see Section 3.2.1), and the other outputs roughly a fifth of that in the stronger Markov model of side information [13], which can be informally understood as a model in which the inputs are independent only when conditioned on the adversary's side information (see Section 3.2.2).

- The Dodis et al. strong two-source extractor [17], with implementation based on [18]. It requires equal length inputs $n_1 = n_2 = n$, where n is a prime with primitive root 2 and outputs $m = \lfloor k_1 + k_2 - n - 2\log_2(\frac{1}{\epsilon})\rfloor$. Its quantum-proof version in the Markov model outputs $m = \lfloor \frac{1}{5}(k_1 + k_2 - n + 8\log_2\epsilon + 9 - 4\log_2(3))\rfloor$.

- The Toeplitz strong seeded extractors based on [20]. It requires a seed length of $n_2 = n_1 + m - 1$ and outputs $m = \lfloor k_1 - 2\log_2(\frac{1}{\epsilon})\rfloor$, when classical-proof or quantum-proof with the same output length. We also offer two-source extensions of this extractor, whereby, the output becomes $m = \lfloor \frac{1}{2}(k_1 + k_2 - n_1 + 1 - 2\log_2(\frac{1}{\epsilon}))\rfloor$ against product quantum side information and a third of that in the Markov model.

- The Trevisan strong seeded extractor [8], with implementation based on [21]. It requires a seed length of $n_2 = O(\log(n_1))$ (see the exact statement of the seed length in Section 4.4) and outputs $m = \lfloor k_1 - 6 - 4\log_2(\frac{m}{\epsilon})\rfloor$ when classical-proof or quantum-proof with the same output length. As with Circulant and Toeplitz, we offer two-source extensions, with the details deferred to Section 4.4.

- The Von Neumann deterministic extractor [36]. Although this extractor does not require a seed, the weak input must have more structure than min-entropy. Additionally, this extractor incurs substantial entropy loss (see Section 4.5 for details). More precisely, it requires that the weak input forms an *exchangeable* sequence, e.g. a suitable weak input is one with IID bits.

Our implementations of the Circulant, Dodis et al. and Toeplitz extractors all have $O(n_1\log(n_1))$ computation time and use the NTT. For these extractors, our code can tolerate input lengths $n_1 \leq 2^{40} \approx 10^{12}$ bits – with an improved throughput version for any input lengths under 2^{29} (for more

details, see Section 2.3). Our implementation of the Von Neumann extractor has a computation time of $O(n_1)$, while Trevisan has a computation time of $O\left(n_1 m \log^2\left(\frac{n_1 m}{\epsilon}\right)\right)$.

2.2 A simple user's guide

To help using `Cryptomite`, Figure 2 presents a flowchart which assists a user on deciding which extractor to use for their application and Figure 3 summarises the achievable parameters. The flowchart is somewhat informal and one can obtain small improvements by analysing each extractor individually (for example, finding a slightly longer output length or shorter seed length). The objective is to give a simple yet good choice for any application, in a clear and easy to follow diagram. In this guide, the user begins with some extractor input of length n_1 and min-entropy k_1 (Definition 1). This flowchart is also implemented as a utility function `suggest_extractor` in `Cryptomite`.

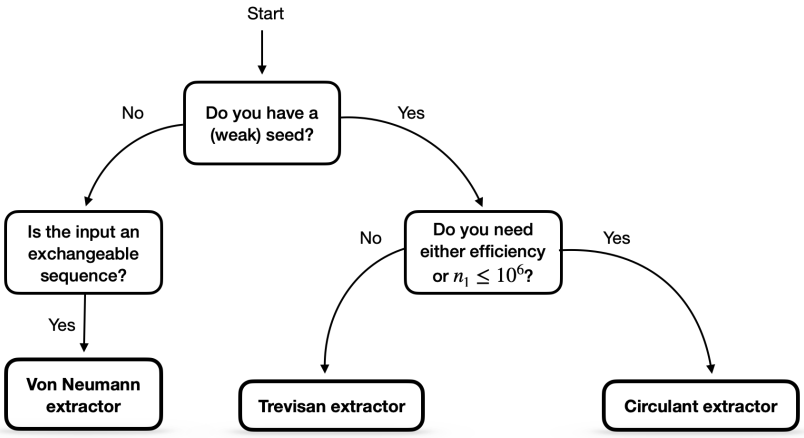

Figure 2: This flow chart shows the simplest way to get (approximately) the most output randomness from an input string of min-entropy k_1 and length n_1, whilst minimising the other resources (seed length and computation time). Small improvements can be obtained by considering Figure 3 below, see for example a discussion on the seed lengths in Appendix A.1. In the left branch, the weak input bits need to form an exchangeable sequence, for which IID bits are a particular case.

			Output size m		
Extractor	**Seed size n_2**	**Computation time**	**Quantum-proof two-source Markov model**	**Quantum-proof seeded ($k_2 = n_2$)**	**Classical-proof two-source Product source model**
Circulant	$n_1 + 1 \in \mathbb{P}$	$O(n_1 \log(n_1))$	$k_1 + (k_2 - n_2) + 2\log(\epsilon)$	$k_1 + 2\log(\epsilon)$	$k_1 + (k_2 - n_2) + 2\log(\epsilon)$
Dodis et al.	$n_1 \in \mathbb{N}_A$	$O(n_1 \log(n_1))$	$\frac{1}{5}(k_1 + (k_2 - n_2) + 8\log(\epsilon) + 4\log(\frac{4}{3}) + 1)$	$\frac{1}{5}(k_1 + 8\log(\epsilon) + 4\log(\frac{4}{3}) + 1)$	$k_1 + (k_2 - n_2) + 2\log(\epsilon) + 1$
Toeplitz	$n_1 + m - 1$	$O(n_1 \log(n_1))$	$k_1 + (k_2 - n_2) + 2\log(\epsilon)$	$k_1 + 2\log(\epsilon)$	$k_1 + (k_2 - n_2) + 2\log(\epsilon)$
Trevisan	$O(\log(n_1))$	$O(n_1 m \log\log(\frac{n_1 m}{\epsilon}))$	$\frac{1}{10}(k_1 + 4(k_2 - n_2) - 4\log(m) + 18\log(\epsilon) + 9\log(\frac{4}{3}) - 6)$	$k_1 - 4\log(m) + 4\log(\epsilon) - 6$	$k_1 + 4(k_2 - n_2) - 4\log(m) + 4\log(\epsilon) - 6$
Von Neumann	-	$O(n_1)$	$\approx^{IID} n(2^{-\frac{k_1}{n}} - 2^{-\frac{2k_1}{n}})$		

Figure 3: A summary of the parameters achievable by the different randomness extractors in `Cryptomite`. For seeded extractors, one uses that $k_2 = n_2$ and all logarithms are in base 2. $\mathbb{P}$ is the set of prime numbers, and $\mathbb{N}_A$ the set of prime numbers with 2 as a primitive root (see Section 4.2). The product source model refers to the two sources being independent (see Section 3.2.1) whilst the Markov model allows for the two sources to be correlated through a common cause (see Section 3.2.2). The parameters for classical side information in the Markov model are not included, but can be computed from Theorem 3. We note that both Circulant and Toeplitz as two-source extractors are secure in the Markov model without a penalty (using Corollary 1), whilst Trevisan and Dodis require using the generic extension of Theorem 4. We have also added a discussion about the seed lengths of the different extractors in practice in Appendix A.1. Finally, the output length for Von Neumann is probabilistic and given in the case of IID bits as input (denoted $\approx^{IID}$).

2.3 Performance

To demonstrate the capabilities of `Cryptomite`, we perform some benchmarking of our constructions on a Apple M2 Max with 64GB RAM processor. The *throughput* (output bits per second) of the extractors of `Cryptomite` are shown in Figure 4. It is calculated by averaging the run-time over 10 trials, for each input length, for input lengths up to 5×10^8. Note that when the input length is below $2^{29} > 5 \times 10^8$ bits, the convolution-based extractors (Circulant, Dodis et al. and Toeplitz) perform operations over a finite field defined by the prime $p = 3 \times 2^{30} + 1$. For input lengths above 2^{29} and up to 2^{40}, the operations must be implemented in a larger finite field and we use the prime $p' = 9 \times 2^{42} + 1$ (which we call `bigNTT`). This change comes at the cost of approximately a 3-4× reduction in throughput.

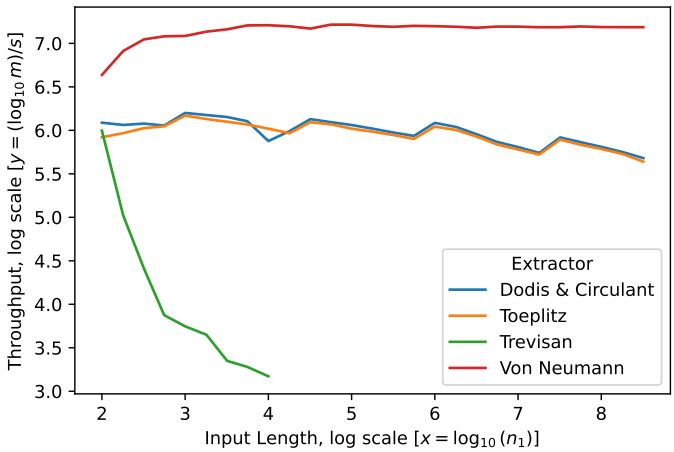

Figure 4: A plot displaying the throughput of our extractor algorithms as a function of the input length (n_1) for each extractor of `Cryptomite`, in logarithmic scale. The throughput is calculated by fixing the input min-entropy to $k_1 = \frac{n_1}{2}$ and calculating the maximum output length according to Figure 3. The necessary seed length for each extractor is given in Section 4.

Some observations are:

- The Circulant, Dodis et al. and Toeplitz extractors are able to output at speeds of up to $\approx$ 1Mbit/s, even for large input lengths. The slight gradual decline in the larger input lengths is expected to be due to CPU characteristics such as cache hit rate and branch prediction.
- The Trevisan extractor can generate output at speeds comparable to the Circulant, Dodis et al. and Toeplitz extractors only when the input or output length is (very) short. For input lengths greater than $n_1 \approx 10^4$, it could not generate a non-trivial throughput.
- The throughput of the Von Neumann extractor is lower for short input lengths due to a constant time overhead.

We note that, for the Circulant, Dodis et al. and Toeplitz extractors, the run-time is independent of the output length (m) due to the NTT implementation step handling a square $n_1 \times n_1$ matrix before outputting the first m bits. This means that the throughput in Figure 4 could be doubled for those extractors if the input had min-entropy has been fixed to a value $k_1 \sim n_1$.

The code used to generate Figure 4 can be found at https://github.com/CQCL/cryptomite/blob/main/bench/bench.py, enabling users to evaluate the performance of the extractors from `Cryptomite` on their specific device.

3 Theory of randomness extraction

In this section, we introduce the notation and relevant definitions that we use for the remainder of the manuscript and provide some useful lemmas and theorems related to randomness extraction (which may be of independent interest). The reader who is only interested in using our extractors can skip the preliminaries and go straight to the section for their extractor of choice and take the parameters directly, or use the repository documentation and parameter calculation modules that we make available.

3.1 Preliminaries

We denote random variables using upper case, e.g. X, which take values x in some finite alphabet with probability $\Pr(X = x) = p_X(x)$, and for some random variable E taking values e, we write the conditional and joint probabilities as $\Pr(X = x|E = e) = p_{X|E=e}(x)$ and $\Pr(X = x, E = e) = p_{X,E}(x, e)$ respectively. We say that two variables X and Y are independent if they are statistically independent, i.e. that $p_{X,Y}(x, y) = p_X(x)p_Y(y)$. We call the probability $p_{\text{guess}}(X) = \max_x p_X(x)$ the (maximum) *guessing probability* and the (maximum) conditional guessing probability $p_{\text{guess}}(X|E) = \max_{x,e} p_{X|E=e}(x)$ (for more detailed discussions and explanations, see [37]). In this work, we focus on the case that the extractor inputs are bit strings, e.g. $X \in \{0,1\}^n$ is a random variable of n bits, and $X = x$ is its realisation. Since the realisations x are bit strings, we denote the individual bits with a subscript, e.g. $x = x_0, \ldots, x_{n-1}$ where $x_i \in \{0,1\}$ for all $i \in 0, \ldots, n-1$. Let $[,]$ denote the concatenation of random variables, for example, if $X \in \{0,1\}^{n_1}$ and $Y \in \{0,1\}^{n_2}$, then $[X, Y] \in \{0,1\}^{n_1+n_2}$ is the concatenation of X and Y. Throughout the manuscript, all logarithms are taken in base 2 and all operations are performed modulo 2, unless explicitly stated otherwise.

The concept of weak randomness, as we consider it, is captured by the (conditional) min-entropy. This can be interpreted as the minimum amount of random bits in a random variable, conditioned on the side information that a hypothetical adversary might possess.

Definition 1 (Conditional min-entropy)**.** The (conditional) min-entropy k of a random variable $X \in \{0,1\}^n$, denoted $\mathrm{H}_\infty(X|E)$, is defined as

$$k = \mathrm{H}_\infty(X|E) = -\log_2 \left(p_{\text{guess}}(X|E)\right) \tag{1}$$

where E is a random variable that contains any additional (or side) information a hypothetical adversary may have to help predict X. The total (conditional) min-entropy divided by its length is the *min-entropy rate* $\alpha = \frac{k}{n}$

In order for randomness extractors to be useful in a cryptographic application, their output must satisfy a *universally composable* [16] security condition. This condition is best understood by imagining a hypothetical game in which a computationally unbounded adversary is given both the *real* output of the extractor and the output of an *ideal* random source. The adversary wins the game if they can *distinguish* between these two bit strings, i.e. guess which of the bit strings came from the extractor versus from the ideal random source. A trivial adversarial strategy is to guess at random, giving a success probability of 1/2 and, for a secure protocol, we ask that a computationally unbounded adversary can not distinguish the bit strings with success probability greater than $\frac{1}{2} + p_{\text{dist}}$, for $p_{\text{dist}} \ll 1$. In other words, the best strategy, even for a very powerful adversary, is not much better than the trivial one. This definition of security is formalised using distance measures as follows.

Definition 2 (Statistical distance)**.** The statistical distance Δ between two random variables X and Y, with $X, Y \in \{0,1\}^n$, conditioned on information E (a random variable of arbitrary dimension), is given by

$$\Delta(X, Y|E) = \frac{1}{2} \max_e \sum_x |p_{X|E=e}(x) - p_{Y|E=e}(x)|. \tag{2}$$

If X and Y satisfy $\Delta(X, Y|E) \leq \epsilon$, we then say that they are ϵ-close (given E), which implies that they can be distinguished with advantage $p_{\text{dist}} \leq \frac{\epsilon}{2}$. This allows us to quantify how close a distribution is to uniform, which gives rise to our term (near-)perfect randomness, where near- is quantified by the distance bound.

Definition 3 (Classical-proof (near-)perfect randomness)**.** The random variable $Z \in \{0,1\}^m$ is (near-)perfectly random (given E), quantified by some value $\epsilon \in (0,1)$, if it is ϵ-close to the uniform distribution U_m over $\{0,1\}^m$, i.e.

$$\Delta(Z, \mathrm{U}_m|E) \leq \epsilon. \tag{3}$$

In our context, ϵ is called the extractor error. In practice, ϵ is a fixed security parameter, for example, enforcing that $\epsilon = 2^{-32}$.

We call the definitions that we have given so far *classical-proof*, in the sense that the adversary's information is classical (a random variable E). When the adversary has access to quantum states to store information, there are situations in which security cannot be guaranteed with Definition 3 alone (see [11] for a concrete example and [38] for a general discussion). Because of this, in what follows we generalise our definitions to the quantum case and say that these are secure against quantum adversaries, or *quantum-proof*.

A random variable $X \in \{0,1\}^n$ with $p_X(x)$ can be written as a quantum state living in the Hilbert space $\mathcal{X}$ as $\rho_{\mathcal{X}} = \sum_{x\in\{0,1\}^n} p_X(x) \, |x\rangle\langle x|$, where the vectors $\{|x\rangle\}_x$ form an orthonormal basis. Considering quantum adversaries (with quantum side information), the classical variable X can be considered part of a composite system potentially correlating it with the adversary in a Hilbert space $\mathcal{X} \otimes \mathcal{E}$, which can be written as the classical-quantum state

$$\rho_{\mathcal{X}\mathcal{E}} = \sum_{x\in\{0,1\}^n} p_X(x) \, |x\rangle\langle x|_{\mathcal{X}} \otimes \rho_{\mathcal{E}|x}. \tag{4}$$

In Equation (4) the adversary's part of the system is the state $\rho_{\mathcal{E}|x}$, which is conditioned on the realisation $X = x$. In the quantum case, the conditional min-entropy (given $\mathcal{E}$) of the classical variable X is $\mathrm{H}_\infty(X|\mathcal{E}) = -\log_2(p_{\text{guess}}(X|\mathcal{E}))$, where $p_{guess}(X|\mathcal{E}) = \max_{\{M_x\}_x} \sum_x \mathrm{Tr}\left(M_x \rho_{\mathcal{E}|x}\right)$ is the adversary's maximum guessing probability to guess X, given all measurement strategies M_x on their

system $\rho_{\mathcal{E}|x}$. For a general discussions on min-entropy in the presence of quantum adversaries, see [39].

Similarly to the classical case, an extractor ideally outputs the classical-quantum state $\mathrm{u}_{\mathcal{Z}} \otimes \rho_{\mathcal{E}}$, where $\mathrm{u}_{\mathcal{Z}}$ is the maximally mixed state and independent of the state of the adversary

$$\mathrm{u}_{\mathcal{Z}} \otimes \rho_{\mathcal{E}} = 2^{-m} \sum_{z \in \{0,1\}^m} |z\rangle\langle z|_{\mathcal{Z}} \otimes \rho_{\mathcal{E}}. \tag{5}$$

Analogously to the classical case, the quantum-proof security criterion is that the randomness extraction process produces a real state (Equation (4) with Z as the classical subsystem) that is essentially indistinguishable from the ideal state in Equation (5). Intuitively, such an ideal state promises that the output $\mathbf{z}$ remains unpredictable for all adversarial strategies using their state $\rho_{\mathcal{E}|z}$. This criterion for security against an adversary who has access to quantum side information is formalised in the following definition.

Definition 4 (Quantum-proof (near-)perfect randomness)**.** The extractor output $Z \in \{0,1\}^m$ is quantum-proof ϵ-perfectly random if

$$\Delta_Q(Z, \mathrm{U}_m|\mathcal{E}) = \frac{1}{2}\|\rho_{\mathcal{Z}\mathcal{E}} - \mathrm{u}_{\mathcal{Z}} \otimes \rho_{\mathcal{E}}\|_1 \leq \epsilon \tag{6}$$

where $\|\cdot\|_1$ denotes the trace norm, defined as $\|\sigma\|_1 = \mathrm{Tr}(\sqrt{\sigma^\dagger \sigma})$. We use the subscript Q and the conditioning on the quantum state $\mathcal{E}$ to mark the distinction with the classical version $\Delta(Z, \mathrm{U}_m|E)$.

For a discussion on why this is the correct measure against quantum adversaries, see [19] 'Section 2'.

We can now define what deterministic, seeded and two-source extractors are at the function level. For simplicity of notation, we omit the conditioning on the side information E (or $\mathcal{E}$), for example, writing $\mathrm{H}_\infty(X)$ instead of $\mathrm{H}_\infty(X|E)$.

Definition 5 (Deterministic randomness extractor)**.** A (ϵ, n, m)-deterministic extractor is a function

$$\mathsf{Ext_d} : \{0,1\}^n \to \{0,1\}^m \tag{7}$$

that maps a random bit string $X \in \{0,1\}^n$ with some specific properties[5], to a new variable $Z = \mathsf{Ext_d}(X)$ that is (optionally quantum-proof) ϵ-perfectly random, see Definition 3 (optionally Definition 4).

Definition 6 (Two-source randomness extractor)**.** Given two independent random variables $X \in \{0,1\}^{n_1}$ and $Y \in \{0,1\}^{n_2}$, with min-entropy $\mathrm{H}_\infty(X) \geq k_1$ and $\mathrm{H}_\infty(Y) \geq k_2$ respectively, a $(n_1, k_1, n_2, k_2, m, \epsilon)$-two-source randomness extractor is a function

$$\mathsf{Ext_2} : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \to \{0,1\}^m \tag{8}$$

where $\mathsf{Ext_2}(X, Y)$ is ϵ-perfectly random, see Definition 3 (optionally Definition 4).

We discuss the role of side information for two-source extractors in Section 3.2. When discussing two-source extractors, we will refer to the input X as the *extractor input* and Y as the *weak seed*. In the case that Y is already near-perfectly random, we will simply call it the *seed*.

Definition 7 (Seeded randomness extractor)**.** A seeded extractor is a special case $(n_1, k_1, n_2, k_2 = n_2, m, \epsilon)$-two-source randomness extractor, where $k_2 = n_2$.

We have mentioned that our extractors are strong, and now we define this property

[5]Such specific properties are, for example, that bits of X are generated in an IID manner [36] or X is a bit-fixing source [40]. For a review of the properties sufficient for deterministic extraction, see [1].

Definition 8 (Strong randomness extractor)**.** A $(n_1, k_1, n_2, k_2, m, \epsilon)$-randomness extractor $\mathsf{Ext} : X \in \{0,1\}^{n_1} \times Y \in \{0,1\}^{n_2} \to \{0,1\}^m$ is strong in the input Y if,

$$\Delta([\mathsf{Ext}(X,Y),Y],[\mathrm{U_m},Y]) \leq \epsilon \tag{9}$$

or quantum-proof strong in the input Y if

$$\Delta_Q([\mathsf{Ext}(X,Y),Y],[\mathrm{U_m},Y]) \leq \epsilon, \tag{10}$$

where $\mathrm{U_m}$ denotes the uniform distribution over m bits. Informally, this can be understood as the property that the extractor output bits are near-perfect, even conditioned on the input Y.

By convention, we will always assume that the extractor is strong in the (weak) seed, although the role of the input sources can usually be exchanged for two-source extractors. All extractors presented in this work are strong with the same error and output length. Having this additional property leads to some useful consequences:

1. Y may become known to the adversary without compromising the security of the extractor output, i.e. the extractor output remains ϵ-perfectly random.

2. Y can be re-used in multiple rounds of randomness extraction (with additive error ϵ), under the assumption that each of the subsequent weak extractor inputs, say $X_2, X_3, \ldots$, are also independent of Y (see Theorem 5).

3. If Y is near-perfectly random (the case of seeded extractors), it can be concatenated with the extractor output to increase the output length. If Y only has min-entropy, it can be reused as the extractor input to another seeded extractor taking the output of the two-source extractor as seed. This allows to extract roughly all the min-entropy $\mathrm{H}_\infty(Y)$ (see Section 3.2.5).

Option 2 is compatible with options 1 and 3 (they can be achieved simultaneously), however option 3 is not compatible with 1 because one should not use Y in cryptographic applications (such as encryption for example) if it is known to an adversary.

3.2 Useful theorems and results

Next, we reproduce and extend results from other works, which provides flexibility in the extractors' use. First, we see how to account for side information with two-source extractors, where both sources might be correlated to the adversary's information. Second, we explain how to use extractors with a bound on the smooth min-entropy[6] only and how to account for near-perfect seeds (which can also be reused in multiple extraction rounds). Third, we see how to generically extend a seeded extractor to a two-source one, as well as how to manipulate the input lengths to construct a family of extractors that can be optimised over. Finally, as mentioned above, we explain how one can concatenate two-source and seeded extractors in order to obtain more advantageous constructions.

3.2.1 Two-source extractors in the product source model

Seeded randomness extractors are well studied in the presence of side information and quantum-proof constructions with minimal (or near minimal) entropy loss are known to exist [7, 41–43], where the weak input to the extractor is in a classical-quantum state, see Equation (4). Multi-source extractors, including two-source extractors require more care, as the adversary might now possess side information about both sources (for example potentially correlating the sources).

Hayashi and Tsurumaru [7] showed that any strong classical-proof seeded extractor has a two-source extension, whereby the extractor can be used with a non-uniform seed at the cost of a larger

[6]The smooth min-entropy $\mathrm{H}_\infty^\delta(X)$ is the maximum min-entropy of any random variable that is δ close (in terms of statistical distance) to the random variable X. For classical-quantum states, the closeness is in terms of trace distance.

error or shorter output length. Moreover, they give an improved extension when the extractor is based on two-universal families of hash functions[7].

Theorem 1 (Classical-proof two-source extension, Theorem 6 and 7 in [7])**.** Any strong classical-proof (or quantum-proof) $(n_1, k_1, n_2, k_2, m, \epsilon)$-seeded extractor is a strong classical-proof (or quantum-proof) $(n_1, k_1, n_2, k_2, m, 2^{n_2-k_2}\epsilon)$-two-source extractor, strong in the (now) weak seed. If the extractor is constructed from a two-universal family of hash functions, it is a strong classical-proof (or quantum-proof) $(n_1, k_1, n_2, k_2, m, 2^{(n_2-k_2)/2}\epsilon)$-two-source extractor.

The quantum-proof claim in Theorem 1 does not directly cover quantum side information about the (now) weak seed, only on the weak input, and still demands that the weak seed and weak input are independent. Using the proof techniques of [46] ('Proof of proposition 1') this can be directly generalised to the case that an adversary has quantum side information about both the input and the weak seed, but with the constraint that, for extractors based on two-universal hash families, this side information is in product form. More concretely, this is the case when the input and the now weak seed form a so-called classical-classical quantum-quantum state, whereby the combined two-universal hashing based extractor input (for the input and (weak) seed given by Equation (4)) takes the form

$$\rho_{XY\mathcal{E}_1\mathcal{E}_2} = \rho_{X\mathcal{E}_1} \otimes \rho_{Y\mathcal{E}_2}. \tag{11}$$

This model for quantum-proof extractors is the *product source model*.

Theorem 2 (Quantum-proof two-source extension in the product source model, Theorem 1 with Proposition 1 in [46])**.** Any strong quantum-proof $(n_1, k_1, n_2, k_2 = n_2, m, \epsilon)$-seeded extractor based on two-universal hash families is a strong quantum-proof $(n_1, k_1, n_2, k_2, m, 2^{(n_2-k_2)/2}\epsilon)$-two-source extractor in the product source model.

Proof. The proof of this follows directly from [46] 'Proof of proposition 1' by noticing that $n-1$ can be generically replaced with n_2, the length of the (weak) seed and that their bound on $(*)$ (as appearing in the text) is found by bounding the collision entropy of the seeded version of the extractor, which holds for all extractors based off two-universal families of hash functions. □

3.2.2 Two-source extractors in the Markov model

Two-source extractors can also be made secure in the so-called *Markov model* [13], which considers stronger adversaries than in the product source model. Instead of requiring the standard independence relation between the input and weak seed, i.e. that the mutual information $\mathrm{I}(X : Y) = 0$, one can use randomness extractors in the presence of side information that correlates the inputs by taking a penalty on the error or output length. The authors prove their results in the general cases of multi-source extractors, so we restate their theorems in the special case of two-source extractors.

In the case of classical side information E, with $\mathrm{I}(X : Y|E) = 0$ instead of $\mathrm{I}(X : Y) = 0$ and with conditional min-entropy's $\mathrm{H}_\infty(X|E) \geq k_1$ and $\mathrm{H}_\infty(Y|E) \geq k_2$;

Theorem 3 (Classical-proof in the Markov model, Theorem 1 in [13])**.** Any (strong) $(n_1, k_1, n_2, k_2, m, \epsilon)$-two-source extractor is a (strong) classical-proof $(n_1, k_1 - \log_2(1/\epsilon), n_2, k_2 - \log_2(1/\epsilon), m, 3\epsilon)$-two-source randomness extractor in the Markov model.

In the quantum case (where the quantum side information of the adversary is now a quantum system $\mathcal{E}$) one requires $\mathrm{I}(X : Y|\mathcal{E}) = 0$ instead of $\mathrm{I}(X : Y) = 0$ and that both sources have conditional min-entropy, i.e. $\mathrm{H}_\infty(X|\mathcal{E}) \geq k_1$ and $\mathrm{H}_\infty(Y|\mathcal{E}) \geq k_2$.

[7] A family $\mathcal{F}$ of functions from $\mathcal{X} \in \{0,1\}^n$ to $\mathcal{Z} \in \{0,1\}^m$ is said to be *two-universal* if $\Pr_f(f(x) = f(x')) \leq \frac{1}{m}$ for any $x \neq x' \in \mathcal{X}$ and f chosen uniformly at random from $\mathcal{F}$ (which is the role of the seed). This definition can be extended beyond picking f uniformly. See [19] (section 5.4 and following) for more details and [44] or the more recent [45] that have shown that two-universal families of hash functions are good randomness extractors.

Theorem 4 (Quantum-proof in the Markov model, Theorem 2 in [13])**.** Any (strong) $(n_1, k_1, n_2, k_2, m, \epsilon)$-two-source extractor is a (strong) quantum-proof $(n_1, k_1 - \log_2(1/\epsilon), n_2, k_2 - \log_2(1/\epsilon), m, \sqrt{3\epsilon 2^{m-2}})$-two-source randomness extractor in the Markov model.

We now show that quantum-proof seeded extractors that are based on two-universal hash families can be transformed into two-source extractors in the Markov model with minimal penalty (only due to having a second weak source). This applies to both our Circulant construction and Toeplitz.

Corollary 1. Any (strong) quantum-proof $(n_1, k_1, n_2, k_2 = n_2, m, \epsilon)$-seeded extractor based on two-universal hash families is also a (strong) quantum-proof $(n_1, k_1, n_2, k_2, m, 2^{(n_2-k_2)/2}\epsilon)$-two-source extractor in the Markov model.

Proof. Via Theorem 2, the quantum-proof $(n_1, k_1, n_2, k_2 = n_2, m, \epsilon)$-seeded extractor becomes a quantum-proof $(n_1, k_1, n_2, k_2, m, 2^{(n_2-k_2)/2}\epsilon)$ two-source extractor in the product source model, as it is based on two-universal hash families. According to [47], a Markov state $\rho_{XY\mathcal{E}}$ (for two sources X, Y, and the adversary's system $\mathcal{E}$, with the Markov conditions discussed in Theorem 4) can be decomposed into product states (Equation (11)) as

$$\rho_{XY\mathcal{E}} = \bigoplus_t p(t)\rho^t_{X\mathcal{E}^t_X} \otimes \rho^t_{Y\mathcal{E}^t_Y} \tag{12}$$

where t runs over a finite alphabet with probability distribution $p(t)$, and $\mathcal{H}_{\mathcal{E}} = \bigoplus_t \mathcal{H}_{\mathcal{E}^t_X} \otimes \mathcal{H}_{\mathcal{E}^t_Y}$ is the Hilbert space of $\mathcal{E}$ (the quantum system held by the adversary). Using this decomposition, we can rewrite Equation (6) as

$$\begin{aligned}
\left\|\rho_{\mathrm{Ext}(X,Y)\mathcal{E}} - u_{\mathcal{Z}} \otimes \rho_{\mathcal{E}}\right\|_1 &= \left\|\mathrm{Ext} \otimes \mathbb{I}_{\mathcal{E}} \left(\rho_{XY\mathcal{E}}\right) - u_{\mathcal{Z}} \otimes \rho_{\mathcal{E}}\right\|_1 && (13)\\
&= \left\|\mathrm{Ext} \otimes \mathbb{I}_{\mathcal{E}} \left(\bigoplus_t p(t)\rho^t_{X\mathcal{E}^t_X} \otimes \rho^t_{Y\mathcal{E}^t_Y}\right) - u_{\mathcal{Z}} \otimes \left(\bigoplus_t \rho^t_{\mathcal{E}^t_X} \otimes \rho^t_{\mathcal{E}^t_Y}\right)\right\|_1 && (14)\\
&= \sum_t p(t) \left\|\mathrm{Ext} \otimes \mathbb{I}_{\mathcal{E}} \left(\rho^t_{X\mathcal{E}^t_X} \otimes \rho^t_{Y\mathcal{E}^t_Y}\right) - u_{\mathcal{Z}} \otimes \rho^t_{\mathcal{E}^t_X} \otimes \rho^t_{\mathcal{E}^t_Y}\right\|_1 && (15)\\
&\leq \sum_t p(t) 2^{(n_2-k_2)/2}\epsilon = 2^{(n_2-k_2)/2}\epsilon && (16)
\end{aligned}$$

where $\mathrm{Ext}(X, Y)$ denotes the application of the extractor on X, Y, which is also understood as the CPTP map $\mathrm{Ext} \otimes \mathbb{I}_{\mathcal{E}}$ on $\rho_{XY\mathcal{E}}$, and $u_{\mathcal{Z}}$ is the maximally mixed state over the output Z of the extractor. In the last inequality, we have used the fact that the extractor is already quantum-proof in the product source model, meaning that, $\left\|\mathrm{Ext} \otimes \mathbb{I}_{\mathcal{E}} \left(\rho^t_{X\mathcal{E}^t_X} \otimes \rho^t_{Y\mathcal{E}^t_Y}\right) - u_{\mathcal{Z}} \otimes \rho^t_{\mathcal{E}^t_X} \otimes \rho^t_{\mathcal{E}^t_Y}\right\|_1 \leq 2^{(n_2-k_2)/2}\epsilon$ for all t, noting that the ccq-state $\rho^t_{X\mathcal{E}^t_X} \otimes \rho^t_{Y\mathcal{E}^t_Y}$ satisfies $\mathrm{H}_\infty(X|\mathcal{E}) \geq k_1$ and $\mathrm{H}_\infty(Y|\mathcal{E})$ for any t, given $\mathrm{H}_\infty(X|\mathcal{E}) \geq k_1$ and $\mathrm{H}_\infty(Y|\mathcal{E})$ for $\rho_{XY\mathcal{E}}$. □

In both Theorems 3, 4 and Corollary 1, one has the option to lift an extractor to the Markov model whilst maintaining whether it is strong or not (a strong extractor will be strong in the Markov model, a weak one will remain weak). We have indicated this option by putting strong in parenthesis in the statements.

The Markov model is particularly relevant in protocols for randomness amplification [18, 48] and in quantum key distribution (QKD)[8]. Other works have considered different forms of quantum side information, for example, where the adversary is allowed specific operations to build the side information $\mathcal{E}$ [49] or when there is a bound on the dimension of the adversary's side information [50]. However, most of these other forms of side information can be viewed as a particular case of

[8] In quantum key distribution (QKD), if the adversary has some predictive power over the seed that is used in the privacy amplification step, one could also imagine that its actions on the quantum channel correlates the raw key material and the (now weak) seed. Such situations, in which the different sources become correlated through a third variable or system, can be accounted for by working in the Markov model.

the Markov one, making it a useful generic extension to use – see the discussion in [13] (Section 1.2: 'Related work'). In the classical case, [12] explores what happens to the security of two-source extractors when the extractor input and the weak seed can be correlated in some specific models, for example, when they have bounded mutual information $I(X:Y) \leq t$, for some constant t.

3.2.3 Working with near-perfect seeds and smooth min-entropy bounds

Seeded randomness extraction can be performed using seeds that are near-perfectly random only (instead of perfectly random), making it more practical. Strong seeded extractors allow the seed to be reused multiple times, and so we give a theorem that accounts for this seed reuse and we discuss this more in Section 3.2.5.

Theorem 5 (Extractors with near-perfect seeds, Appendix A in [51])**.** Given an ϵ_s-perfect seed Y, which is used to extract t times from a weak input[9] with X_i for $i = 1, ..., t$ such that $\mathrm{H}_\infty(X_i|X_{i-1}, ..., X_1) \geq k$ for all i using a strong $(n_1, k_1 = k, n_2, k_2 = n_2, m, \epsilon)$-seeded extractor $\mathsf{Ext_s}(X, Y)$ each time, the concatenated output $[\mathsf{Ext_s}(X_1, Y), ..., \mathsf{Ext_s}(X_t, Y)]$ is ϵ_t-perfect with

$$\epsilon_t \leq t \cdot \epsilon + \epsilon_s. \tag{17}$$

Similarly, in practice, often only a bound on the smooth min-entropy can be obtained, for example when using the (generalised) entropy accumulation theorem [52–55] or probability estimation factors [56–58]. We give two theorems that allow for seeded and two-source extractors to function with inputs that have a promise on their smooth min-entropy.

Theorem 6 (Seeded extraction with smooth min-entropy, e.g. Corollary 7.8 in [59])**.** Given a (strong) quantum-proof $(n_1, k_1 = k, n_2, k_2 = n_2, m, \epsilon)$-seeded extractor, one can use a weak input X with smooth min-entropy $\mathrm{H}_\infty^\delta(X|\mathcal{E}) \geq k$ at the cost of an additive error $\epsilon + \delta$, i.e. giving a $(n_1, k_1 = k, n_2, k_2 = n_2, m, \epsilon + \delta)$-seeded extractor.

Theorem 7 (Two-source extraction with smooth min-entropy, Lemma 17 in [55])**.** Given a (strong) quantum-proof $(n_1, k_1 - \log_2(1/\epsilon_1) - 1, n_2, k_2 - \log_2(1/\epsilon_2) - 1, m, \epsilon)$-two-source extractor in the Markov model (see Section 3.2.2), one can use weak inputs X and Y satisfying $\mathrm{I}(X:Y|\mathcal{E}) = 0$ with smooth min-entropy's $\mathrm{H}_\infty^{\delta_1}(X|\mathcal{E}) \geq k_1'$ and $\mathrm{H}_\infty^{\delta_2}(Y|\mathcal{E}) \geq k_2'$ giving a

$$\left(n_1, k_1' - \log_2\left(\frac{1}{\epsilon_1}\right), n_2, k_2' - \log_2\left(\frac{1}{\epsilon_2}\right), m, 2\epsilon + 6(\delta_1 + \delta_2) + 2(\epsilon_1 + \epsilon_2)\right) - \text{extractor} \tag{18}$$

This result also applies to classical-proof extractors and, because of the generality of the Markov model, to many other forms of (quantum) side information [13].

3.2.4 Extending extractors to more settings

As some of our extractors require specific input lengths (for example, that the input length must be prime), we give some results that give flexibility to increase or decrease the length of the extractor input, which later allows us to optimise the output length. First, we show that any two-source extractor can be used as a seeded extractor with a short seed, by concatenating the seed with fixed bits.

Theorem 8 (Short seeded extractors from two-source extractors)**.** Any $(n_1, k_1, n_2, k_2, m, \epsilon)$-two-source randomness extractor $\mathsf{Ext_2}$ is a $(n_1, k_1, n_2' = k_2, k_2, m, \epsilon)$-seeded randomness extractor $\mathsf{Ext_s}$ for any $k_2 \leq n_2$, where $\mathsf{Ext_s}$ is constructed by first concatenating the seed (of length k_2) with a fixed bit string of length $n_2 - k_2$ and then applying $\mathsf{Ext_2}$. Specifically, given a seed $Y \in \{0,1\}^{k_2}$, the seeded extractor is constructed from the two-source one as $\mathsf{Ext_s}(X, Y) = \mathsf{Ext_2}(X, [Y, c])$, where $[Y, c]$ denotes the concatenation of the seed Y with any fixed string $c \in \{0,1\}^{n_2 - k_2}$ (e.g. $c = \{0\}^{n_2 - k_2}$).

[9] Called a block min-entropy condition. The theorem can be generalised to the case that the min-entropy of each block differs, i.e. $\mathrm{H}_\infty(X_i|X_{i-1}, ..., X_1) \geq k_i$.

Proof. Let $Y \in \{0,1\}^{k_2}$ be a seed (i.e. it has full min-entropy $\mathrm{H}_\infty(Y) = k_2$) and $c \in \{0,1\}^{n_2-k_2}$ be a fixed string, e.g. $\{0\}^{n_2-k_2}$, with $\mathrm{H}_\infty(c) = 0$. Define Y' as the concatenation of Y and c, $Y' = [Y, c]$, which has min-entropy $\mathrm{H}_\infty(Y') = \mathrm{H}_\infty([Y, c]) = \mathrm{H}_\infty(Y) = k_2$. Since Y' is a random variable with min-entropy k_2 and length n_2, it can be used as the second input to any $(n_1, k_1, n_2, k_2, m, \epsilon)$-two-source extractor $\mathsf{Ext}_2(X, Y')$. Therefore, $\mathsf{Ext}_2(X, Y')$ can be understood as a seeded extractor $\mathsf{Ext}_\mathsf{s}(X, Y)$, constructed by first concatenating the seed Y with a fixed string of bits then applying Ext_2. In other words, $\mathsf{Ext}_\mathsf{s}(X, Y) = \mathsf{Ext}_2(X, [Y, c]) = \mathsf{Ext}_2(X, Y')$. □

This theorem is useful when a seed is too short to be used in a seeded extractor for a given weak input. For example, consider a seed Y of length n_2 and a weak input $X \in \{0,1\}^{n_1}$ with $n_1 > n_2$. If n_1 is sufficiently large compared to n_2, the seed length n_2 may be insufficient to perform extraction using existing seeded extractor constructions. However, by concatenating Y with a zero string of length c, one can create a weak seed Y' of length $n_2 + c$ with the same min-entropy $k_2 = n_2$. This weak seed is now a valid second input for a two-source extractor, which may give more freedom and allow extraction to become possible.

We now show how to shorten the weak input length, if desired.

Lemma 1. For any integer $c \leq n$, a random variable $X \in \{0,1\}^n$ with min-entropy $\mathrm{H}_\infty(X) = k$ can be shortened to a new random variable $X' \in \{0,1\}^{n-c}$ with min-entropy $\mathrm{H}_\infty(X') \geq k - c$, by removing the last c bits of X.

Proof. Let $X' \in \{0,1\}^{n_1-c}$, $\bar{X}' \in \{0,1\}^c$ and $X = [X', \bar{X}'] \in \{0,1\}^{n_1}$, where $[,]$ denotes concatenation. By definition, $\mathrm{H}_\infty(X) = k$ and we want to prove a lower bound on $\mathrm{H}_\infty(X')$. Using that $p_{X'}(x')$ is a marginal distribution of $p_{[X',\bar{X}']}([x', \bar{x}'])$, we have that

$$\mathrm{H}_\infty(X') = -\log_2(\max_{x'} p_{X'}(x')) = -\log_2\left(\max_{x'} \sum_{\bar{x}' \in \{0,1\}^c} p_{[X',\bar{X}']}([x', \bar{x}'])\right). \tag{19}$$

Continuing from Equation (19) by passing the maximum inside the sum and then maximise over both variables (instead of one), we get

$$-\log_2\left(\max_{x'} \sum_{\bar{x}' \in \{0,1\}^c} p_{[X',\bar{X}']}([x', \bar{x}'])\right) \geq -\log_2\left(\sum_{\bar{x}' \in \{0,1\}^c} \max_{x'} p_{[X',\bar{X}']}([x', \bar{x}'])\right) \tag{20}$$

$$\geq -\log_2\left(\sum_{\bar{x}' \in \{0,1\}^c} \max_{[x',\bar{x}']} p_{[X',\bar{X}']}([x', \bar{x}'])\right) = -\log_2\left(\sum_{\bar{x}' \in \{0,1\}^c} \max_x p_X(x)\right) \tag{21}$$

$$\geq -\log_2\left(\sum_{\bar{x}' \in \{0,1\}^c} 2^{-k}\right) = -\log_2(2^{-k}2^c) = k - c \tag{22}$$

where we have used that $\max_x p_X(x) \leq 2^{-k}$, since $\mathrm{H}_\infty(X) = k$ by definition. □

3.2.5 Composing extractors

Extractors can be composed together to form new extractors with improved parameters. One can append a strong seeded extractor to a strong two-source extractor, taking the two-source extractor output as a seed and the two-source weak seed as the extractor input. Doing so, one can construct a $(n_1, k_1, n_2, k_2, m + \max\{k_1, k_2\}, \epsilon_2 + \epsilon_s)$-two-source extractor, which significantly increases the output length. For more details, see Lemma 38 in [13] and the discussion in Section 4.6 of [18].

4 Library in detail

In this section, we define the extractors in `Cryptomite` concretely, giving the implementation details and parameter calculations.

4.1 Our new Circulant construction

We now expose the details of our new extractor construction Circulant.

Definition 9. Let $x = x_0, \ldots, x_{n-2} \in \{0,1\}^{n-1}$ and $y = y_0, \ldots, y_{n-1} \in \{0,1\}^n$ where n is prime. The function $\mathsf{Circulant}(x,y) : \{0,1\}^{n-1} \times \{0,1\}^n \to \{0,1\}^m$ is implemented by:

1. Set $x' = [x, 0] \in \{0,1\}^n$, where $[,]$ denotes concatenation (i.e. $x'_{n-1} = 0$).
2. Then,

$$\mathsf{Circulant}(x,y) = \left(\mathrm{circ}(x')y\right)_{0:m-1}, \tag{23}$$

where the matrix-vector multiplication $\mathrm{circ}(x')y$ is taken mod 2 (in each component of the resulting vector) and the subscript $0 : m-1$ denotes the first m elements of the vector. The term $\mathrm{circ}(x')$ is the $n \times n$ circulant matrix generated by $x' = x'_0, \ldots x'_{n-1}$,

$$\mathrm{circ}(x') = \begin{bmatrix} x'_0 & x'_1 & x'_2 & \ldots & x'_{n-2} & x'_{n-1} \\ x'_{n-1} & x'_0 & x'_1 & x'_2 & \ldots & x'_{n-2} \\ & \ddots & \ddots & \ddots & \ddots & \\ x'_1 & x'_2 & x'_3 & \ldots & x'_{n-1} & x'_0 \end{bmatrix}, \tag{24}$$

Using the result in the Appendix of [18], the function in Definition 9 can be implemented in $O(n \log n)$ computation time.

Theorem 9. The function $\mathsf{Circulant}(X,Y) : \{0,1\}^{n-1} \times \{0,1\}^n \to \{0,1\}^m$ in Definition 9 is a strong in Y, classical-proof and quantum-proof $(n_1 = n-1, k_1, n_2 = n, k_2 = n, m, \epsilon)$-seeded extractor for prime n, with output length

$$m \leq k_1 + 2\log_2(\epsilon). \tag{25}$$

The proof can be found in Appendix A.3.

Using Theorem 1 to build a two-source extension that is (classical-proof and) quantum-proof in the product source model, we get that our Circulant extractor has an output length

$$m \leq k_1 + k_2 - n + 2\log_2(\epsilon) \tag{26}$$

and, by Corollary 1, quantum-proof in the Markov model with the same output length.

4.2 Dodis et al.

The exact Dodis et al. extractor [17] $\mathsf{Dodis\ et\ al.}(x,y) : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m$ construction is given in Appendix A.4, whilst we give its idea here. The extractor input $x = x_0, x_1, \ldots, x_{n-1} \in \{0,1\}^n$, and the (weak) seed $y = y_0, y_1, \ldots, y_{n-1} \in \{0,1\}^n$ must both be of equal length n. The extractor construction uses a set of m $n \times n$ matrices, which we label $A_0, ..., A_{m-1}$, with entries in $\{0,1\}$ (i.e. bits), which must be chosen such that, for any subset $B \subseteq A_0, ..., A_{m-1}$, the sum of the matrices in the subset B has rank at least $n - r$ for some small constant r (which acts as a penalty to the output length). The extractor output of m bits is then

$$\mathsf{Dodis\ et\ al.}(x,y) = [(A_0 x) \cdot y, (A_1 x) \cdot y, ..., (A_{m-1} x) \cdot y] \tag{27}$$

where $\cdot$ is the inner product modulo 2. We give such a set of matrices, based on "Cyclic Shift matrices" from [17] Section 3.2, and a version of Equation (27) that can be implemented in $O(n \log(n))$ computation time, based on [18] in Appendix A.4. This construction requires that $x \neq \{0\}^n$ or $\{1\}^n$

and that n is a prime with 2 as a primitive root[10]. For the input lengths considered in this work (i.e. up to 10^{12}), this set is sufficiently dense in the natural numbers for all practical purposes. We also note that fixing the input length is a pre-computation that can be done very efficiently.

We note that the particular choice of Cyclic Shift matrices $\{A_i\}_i$, together with the padding of step 1 in Definition 9, gives the Circulant construction – which we prove to be a two-universal family of hash functions and in turn use known proof techniques and results, for example, giving that Circulant is quantum-proof without a penalty [19].

Theorem 10 (Classical-proof, from [17])**.** Our implementation of the Dodis et al. extractor is a strong classical-proof $(n_1 = n, k_1, n_2 = n, k_2 = n, m, \epsilon)$-seeded randomness extractor with

$$m \leq k_1 + 1 + 2\log_2 \epsilon, \tag{28}$$

and a classical-proof $(n_1 = n, k_1, n_2 = n, k_2, m, \epsilon)$-two-source randomness extractor, strong in the weak seed (with min-entropy k_2), with

$$m \leq k_1 + k_2 - n + 1 + 2\log_2 \epsilon. \tag{29}$$

We note that the Dodis et al. extractor was shown to be a two-source extractor directly [17], i.e. without using a generic extension, and therefore outputs one more bit than other constructions in this setting. Contrary to the Circulant construction, Dodis et al. is not known to be quantum-proof directly (i.e. without a penalty) even as a seeded extractor. Therefore, as explained in Section 3.2.2, we make it quantum-proof in the Markov model [13] by taking a penalty on the output length.

Theorem 11 (Quantum-proof in the Markov model, Proposition 5 of Appendix B in [18])**.** The Dodis et al. extractor becomes a strong quantum-proof (in the Markov model) $(n_1 = n, k_1, n_2 = n, k_2 = n, m, \epsilon)$-seeded randomness extractor, with

$$m \leq \frac{1}{5}\left(k_1 + 8\log_2 \epsilon + 4\log_2\left(\frac{4}{3}\right) + 1\right), \tag{30}$$

and a strong quantum-proof (in the Markov model) $(n_1 = n, k_1, n_2 = n, k_2, m, \epsilon)$-two-source randomness extractor, where

$$m \leq \frac{1}{5}\left(k_1 + k_2 - n + 8\log_2 \epsilon + 4\log_2\left(\frac{4}{3}\right) + 1\right). \tag{31}$$

4.3 Toeplitz

The Toeplitz extractor [20] $\mathsf{Toeplitz}(x, y) : \{0,1\}^n \times \{0,1\}^{n+m-1} \to \{0,1\}^m$ is constructed using Toeplitz matrices. Given $x = x_0, x_1, \ldots x_{n-1} \in \{0,1\}^n$ and $y = y_0, y_1, \ldots y_{n+m-2} \in \{0,1\}^{n+m-1}$, the output of the extractor is computed as the matrix-vector multiplication

$$\mathsf{Toeplitz}(x, y) = \mathrm{toep}(y)x \tag{32}$$

where

$$\mathrm{toep}(y) = \begin{bmatrix} y_0 & y_{n+m-2} & \cdots & \cdots & y_m \\ y_1 & y_0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ y_{m-1} & y_{m-2} & \cdots & \cdots & . \end{bmatrix} \tag{33}$$

is the $n \times m$ Toeplitz matrix generated from y. The full implementation details, in computation time $O(n\log(n))$, is given in Appendix A.5. As it was shown to be directly quantum-proof [19, 60], we state the claims as a single theorem.

[10] A list of all primes with 2 as a primitive root up to 10^6 can be found at https://github.com/CQCL/cryptomite/blob/main/na_set.txt. We also provide a function which allows the user to compute the closest prime with 2 as primitive root, as well as individually the closest above and below the input length, see the utility functions https://github.com/CQCL/cryptomite/blob/main/cryptomite/utils.py.

Theorem 12. The Toeplitz extractor is a strong classical- and quantum-proof $(n_1 = n, k_1, n_2 = n + m - 1, k_2 = n + m - 1, m, \epsilon)$-seeded randomness extractor, where

$$m \leq k_1 + 2\log_2(\epsilon), \tag{34}$$

Using the two weak sources extension (Theorem 1), a classical- and quantum-proof $(n_1 = n, k_1, n_2 = n + m - 1, k_2, m, \epsilon)$-two-source randomness extractor strong in the weak seed (with min-entropy k_2), where

$$m \leq \frac{1}{2}(k_1 + k_2 - n + 1 + 2\log_2(\epsilon)), \tag{35}$$

in the product source model and also in the Markov model (by Corollary 1). Note that k_2 may be bigger than n, due to the fact that the length of the (weak) seed is $n + m - 1$.

Modified Toeplitz Extractor – To reduce the seed length, one can use modified Toeplitz matrices (Subsection B-B in [7]), which allow a reduction from $n + m - 1$ to $n - 1$. The authors also prove the security of the two-source extension against quantum side information on the weak input only (which we reproduced as Theorem 1) and in [46] the security is extended against product quantum side information (i.e. side information on both inputs). One could further extend it to allow for more general forms of quantum side information (in the Markov model) using Theorem 4.

4.4 Trevisan

The Trevisan extractor [8] $\mathsf{Trevisan}(x, y) : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is important because of its asymptotic logarithmic seed length $d = O(\log(n))$. At a high level, it is built from two components: (1) a weak design which expands the uniform seed into several separate bit strings (*chunks*) that have limited overlap, in the sense that only a subset of bits from any two strings match, and (2) a 1-bit extractor which combines each of the chunks, in turn, with the weak input to produce one (near-)perfect bit. By applying the 1-bit extractor to each expanded chunk, the Trevisan extractor produces multiple bits of near-perfect randomness. Its drawback is its computation time, which often prohibits its use in practice. The implementation, based on the building blocks of [21], in computation time $O(n^2 \log\left(\log\left(\frac{n^2}{\epsilon}\right)\right))$, can be found in Appendix A.6 and has a seed length, d, of

$$d = at^2, \tag{36}$$

with

$$t \in \mathbb{P}_{\geq q} \quad : \quad q = 2\lceil \log_2(n) + 2\log_2(2m/\epsilon) \rceil, \tag{37}$$

$$a = \max\left\{1, \left\lceil \frac{\log_2(m - 2\exp(1)) - \log_2(t - 2\exp(1))}{\log_2(2\exp(1)) - \log_2(2\exp(1) - 1)} \right\rceil\right\}, \tag{38}$$

where $\mathbb{P}_{\geq q}$ denotes the set of primes that are larger or equal to q and $\exp(1) \approx 2.718$ is the basis of the natural logarithm. As for Circulant and Toeplitz, Trevisan is directly quantum-proof [43], hence we state the claims as one theorem only.

Theorem 13 (Combining parameters [21], with quantum-proof in [43])**.** The Trevisan extractor is a strong (classical-proof or) quantum-proof $(n_1 = n, k_1, n_2 = d, k_2 = d, m, \epsilon)$-seeded randomness extractor, with

$$m \leq k_1 + 4\log_2(\epsilon) - 4\log_2(m) - 6. \tag{39}$$

Using the two-weak-source extension (Theorem 1), the Trevisan extractor is a classical-proof $(n_1 = n, k_1, n_2 = d, k_2, m, \epsilon)$-two-source randomness extractor strong in the weak seed (with min-entropy k_2), where

$$m \leq k_1 + 4k_2 - 4d + 4\log_2(\epsilon) - 4\log_2(m) - 6, \tag{40}$$

and, using Theorem 4, a quantum-proof $(n_1 = n, k_1, n_2 = d, k_2, m, \epsilon)$-two-source randomness extractor in the Markov model with

$$m \leq \frac{1}{10}\left(k_1 + 4(k_2 - d) - 4\log_2(m) + 18\log_2\epsilon + 9\log_2\left(\frac{4}{3}\right) - 6\right). \tag{41}$$

Trevisan with shorter seed length – Our implementation uses the Block Weak Design from [21] which iteratively calls the weak design of [61] to generate the necessary chunks. Using other weak designs allow for the seed length to be made shorter in practice, at the expense of increasing the entropy loss. For example, one can use our implementation with the weak design of [61] directly, to achieve a seed length $d = t^2$ at the expense of reducing the output by a factor of $m \to m/2\exp(1)$ i.e. giving a minimum entropy loss of at least $1 - 1/2\exp(1) \approx 82\%$.

4.5 Von Neumann

The Von Neumann deterministic extractor is a function $\text{Von Neumann}(x) : \{0,1\}^n \to \{0,1\}^m$ taking a single input $x = x_0, x_1, \ldots x_{n-1} \in \{0,1\}^n$ where the random variables that generate each bit form an *exchangeable sequence*. More concretely, the Von Neumann extractor requires that the input probability distribution of X generating the input x satisfies

$$\Pr(X_{2i} = 0|E) = \Pr(X_{2i+1} = 0|E) = p_i, \tag{42}$$

for all $i \in \{0, \ldots, \lfloor n/2 \rfloor - 1\}$, some $p_i \in [0,1]$ and some adversary side information E (or alternatively, for a quantum-proof extractor, side information $\mathcal{E}$). Let $j = 0, \ldots, \lfloor n/2 \rfloor - 1$, the jth output bit of the Von Neumann extractor is then given by

$$\text{Von Neumann}(\mathbf{x})_j = \begin{cases} x_{2j} & \text{if} \quad x_{2j} \neq x_{2j+1} \\ \emptyset & \text{otherwise} \end{cases}, \tag{43}$$

where the empty set $\emptyset$ denotes that there is no output bit in that round. Note that, contrary to the other constructions, the output length of the Von Neumann extractor is probabilistic. For example, a valid input X is one that is IID with an unknown bias $p_i = p$ for all i. In this case, the approximate (due to finite sampling from X) output length is given by

$$m \approx p(1-p)n, \tag{44}$$

which gives an entropy loss $k - m$ of $(1 + (1-p)p/\log_2(\max\{p, 1-p\}))k_1 \geq 3k_1/10$ i.e. implying at least 30% entropy loss. This is significantly worse than the entropy loss when using the same input to an optimal seeded extractor, where $m = \lfloor -n\log_2(\max\{p, 1-p\}) - 2\log_2(\frac{1}{\epsilon})\rfloor$ i.e. approximately 0% entropy loss when ϵ is constant. Other extractors based on this construction, but optimising the output length by recycling unused bits, allow for this entropy loss to be improved, including the Elias and Peres extractor [62] and the generalised Von Neumann extractor [63]. However, they are still unable to achieve the same entropy loss that is possible with seeded extractors, in general. For completeness, we provide a pseudo-code implementation in Appendix A.7 with computation time $O(n)$.

5 Cryptomite: Examples with code

In this section we showcase `Cryptomite` by giving code examples for privacy amplification in quantum key distribution (QKD) and randomness extraction in random number generation (RNG). We give the Python code specific to the experimental demonstrations in [64, 65], as well as extensions that improve their results by relaxing assumptions (hence increasing security) and reducing resources. We note that one immediate benefit of using our extractors is also the numerical precision obtained by using the NTT for performance. These concrete examples and code can easily be adapted to any setup requiring a randomness extractor. More examples can also be found in the documentation.

Remark that all the examples that we give in this section also showcase the limitation of the Trevisan extractor (and others) in practice. Indeed, for these examples the input lengths are too long for implementations that do not have quasi-linear computation time $O(n\log(n))$, i.e. that the usual notion of polynomial efficiency is irrelevant in practical quantum cryptography. We refer the reader to Figure 4 for a numerical example of the performance with different input lengths.

5.1 `Cryptomite` for privacy amplification in QKD

In QKD, the goal is for two parties, Alice and Bob, to generate a shared secret key that is unpredictable to an adversary. After rounds of quantum and classical communication, Alice and Bob share an identical shared raw key which is only partially secret to the adversary (i.e. after state sharing and measurement, sifting, parameter estimation and error reconciliation). Randomness extractors are used for privacy amplification, a subroutine which transforms the raw key (that has some min-entropy) into a final secret key that is (almost) completely secret to adversary (i.e. ϵ-perfectly random to the adversary). In standard protocols, this task is performed using a strong seeded extractor, which requires both Alice and Bob to have a shared (near-)perfect seed.

We consider the continuous-variable QKD demonstration of [64], which aims at security against quantum adversaries and therefore requires a quantum-proof extractor. From the experiment, Alice and Bob obtain $n = 1.738 \times 10^9$ bits of shared raw key as input for privacy amplification[11]. After evaluating the total min-entropy of the raw key, the authors compute the final secret key length of $m = 41378264$ bits based on their given security parameters (including an extraction error of $\epsilon = 10^{-10}$) when using their Toeplitz extractor implementation. This extraction requires a perfect seed of length $d = m + n - 1 = 1.738 \times 10^9 + 41378263$ and can be performed in a few lines of Python code using `Cryptomite`, as shown in Figure 5.

Extensions – The Circulant extractor can be used to generate the same amount of shared secret key, whilst substantially reducing the length of the seed. The code to perform this is given in Figure 6. The setup can be further improved, as in the experiment a quantum-RNG (QRNG) is used to generate the extractor seed [66] and its quality relies on the correct characterisation and modelling of the components in the device (an assumption which was shown to be potentially problematic in [67]). We show how to relax this assumption by assuming that the min-entropy rate of the QRNG output is only $r = 0.99$, i.e. $k_2 = r \cdot n_2$, and adjusting the output length of the extractor accordingly (in the quantum-proof product source model), see Figure 3. The code to perform the parameter calculation is given in Figure 6, but it could also easily be performed using the calculation module `from_params` (see documentation).

[11] Since the extractor input length is bigger than 2^{29}, the extractor implementation will use the `bigNTT`, implying a throughput of $3-4\times$ less than that shown in Section 2.3.

```python
import cryptomite
def privacy_amplification(raw_key_bits: list, seed_bits: list,
                          n_1 = 1.738 * 10**9, m = 41378264)
    """ Perform privacy amplification for the QKD protocol.

    Parameters
    ----------
    raw_key_bits : list of bits, derived from the measurement
      outcomes (after sifting, error correction and parameter
      estimation).
    seed_bits : list of bits, generated independently.
    n_1: integer, the length of the raw key bit string.
    m: integer, the length of the output secret key bit string.

    Returns
    ---------
    list of bits,
      the extracted output (i.e. the shared secret key).
    """
    # Initialise the Toeplitz extractor with the appropriate parameters:
    toeplitz = cryptomite.Toeplitz(n_1, m)
    # Perform Toeplitz extraction and return the output:
    return toeplitz.extract(raw_key_bits, seed_bits)
```

Figure 5: The implementation of privacy amplification for [64] using the Toeplitz extractor from `Cryptomite`.

```python
import cryptomite
from math import floor, log2
def improved_privacy_amplification(raw_key_bits: list, seed_bits: list,
                                   n_1 = 1.738 * 10**9, epsilon = 10**(-10),
                                   m = 41378264, r = 0.99)
    """ Perform improved privacy amplification for the QKD protocol.

    Parameters
    ----------
    raw_key_bits : list of bits, derived from the measurement outcomes
      (after sifting, error correction and parameter estimation).
    seed_bits : list of bits, generated independently.
    n_1: integer, the length of the raw key bit string.
    epsilon: float, the extractor error.
    m: integer, the length of the output secret key bit string.
    r: float, the min-entropy rate of the extractor (now weak) seed.

    Returns
    ---------
    list of bits, the extracted output (i.e. the shared secret key).
    """
    # Calculate the input min-entropy from the given extractor
    # output length and extractor error:
    k_1 = m - 2*log2(epsilon)
    # To use Circulant, the seed length needs to be a prime, we
    # find the one larger than or equal to n_1 + 1 (see Def. 9):
    n_adjusted = cryptomite.utils.next_prime(n_1 + 1)
    # If n_adjusted > n_1 + 1, we increase the length of
    # the raw key to n_adjusted - 1 by padding 0's to it:
    if n_adjusted > n_1 + 1:
        raw_key_bits += [0]*(n_adjusted - n_1 - 1)
    # Take the correct number of seed bits (less than with Toeplitz):
    seed_bits = seed_bits[:n_adjusted]
    # Compute the new weak seed's min-entropy, given its
    # min-entropy rate r:
    k_2 = r * n_adjusted
    # Compute the extractor's output length (product source
    # model, see Fig. 3):
    m = floor(k_1 + k_2 - n_adjusted + 2*log2(epsilon))
    # Initialise the Circulant extractor with the appropriate parameters:
    circulant = cryptomite.Circulant(n_adjusted - 1, m)
    # Perform extraction and return the output:
    return circulant.extract(raw_key_bits, seed_bits)
```

Figure 6: Improved implementation of privacy amplification for [64] using the Circulant extractor with a weak seed.

5.2 `Cryptomite` for randomness extraction in (Q)RNG

In random number generation (RNG), randomness extraction is used to process the raw output from an entropy source (sometimes called the noise source) into the final output that is ϵ-perfectly random – a process analogous to *conditioning* in the NIST standards [68]. In most protocols for quantum RNG (QRNG), this is done using (strong) seeded randomness extractors. To showcase `Cryptomite` for RNG, we replicate and extend the randomness extraction step of the semi-device-independent QRNG based on heterodyne detection [65], again giving code examples.

In [65], the randomness extractor error is chosen to be $\epsilon = 10^{-10}$ and the length of the extractor input is $n_1 = 6.5 \times 10^9$ bits[12]. The experimental results and calculations certify a min-entropy rate of $k_1/n_1 = 0.08943$ and, therefore, a total min-entropy of $k_1 = \lfloor 0.08943 n_1 \rfloor = 581295000$ for the extractor input. The randomness extraction step is performed using the Toeplitz extractor, which gives an output length of $m = 581294933$ bits and requires a seed length of $n_2 = n_1 + m - 1 = 7081294932$ bits. This can be performed in a few lines of code using `Cryptomite`, as shown in Figure 7.

Extension – As in the QKD example, the RNG case can be improved using other options given in our extractor library, reducing the seed length and allowing the option to relax the requirement on the seed min-entropy rate. Using our Circulant extractor instead of the Toeplitz extractor saves 581294923 seed bits. To implement this, the input length $n_1 = 6.5 \times 10^9$ needs to be adjusted to n_1' such that $n_1' + 1$ is a prime number. Based on this adjustment to n_1, the min-entropy must also be recalculated (using either Theorem 8 or Lemma 1). We find that the closest prime is $6.5 \times 10^9 + 9$, so we set $n_1' = 6.5 \times 10^9 + 8$ and pad the weak input with 8 fixed bits. Moreover, using a seeded extractor for random number generation demands a (near-)perfectly random seed as the resource, which leads to a circularity. As in the QKD example, using `Cryptomite` we introduce a min-entropy rate parameter r that allows the extractor seed to be weakly random only, whilst still outputting near-perfect randomness. The code for this extension can be found in Figure 8.

```
import cryptomite
from math import floor, log2
def randomness_extraction(raw_randomness: list, seed_bits: list,
                          n_1 = 6500000000, epsilon = 10**-10,
                          k_1 = 581295000)
    """ Perform randomness extraction for the RNG protocol.

    Parameters
    ----------
    raw_randomness : list of bits, the outcomes of measurements.
    seed_bits : list of bits, generated independently.
    n_1: integer, the length of the raw randomness bit string.
    epsilon: float, the extractor error.
    k_1: float, the total min-entropy of the raw randomness.

    Returns
    ---------
    list of bits,
      the extracted output (i.e. the near-perfect randomness).
    """
    # Compute the extractor output length:
    m = floor(k_1 + 2*log2(epsilon))
    # Initialise the Toeplitz extractor with the appropriate
    # parameters:
    toeplitz = cryptomite.Toeplitz(n_1, m)
    # Perform Toeplitz extraction and return the output:
    return toeplitz.extract(raw_randomness, seed_bits)
```

Figure 7: The implementation of randomness extraction for [65] using the Toeplitz extractor in `Cryptomite`.

[12] Again, since the extractor input length is bigger than 2^{29}, the extractor implementation will use the `bigNTT`, implying a throughput of $3-4\times$ less than that shown in Section 2.3.

```python
import cryptomite
from math import floor, log2
def improved_randomness_extraction(raw_randomness: list, seed_bits: list,
                                   n_1 = 6500000000, epsilon = 10**-10,
                                   k_1 = 581295000, r = 0.99)
    """ Perform improved randomness extraction for the RNG protocol.

    Parameters
    ----------
    raw_randomness : list of bits, the outcomes of measurements.
    seed_bits : list of bits, generated independently.
    n_1: integer, the length of the raw randomness bit string.
    epsilon: float, the extractor error.
    k_1: float, the total min-entropy of the raw randomness.
    r: float, the min-entropy rate of the extractor (now weak) seed.

    Returns
    ---------
    list of bits,
      the extracted output (i.e. the near-perfect randomness).
    """
    # To use Circulant, the seed length needs to be a prime,
    # we find the one closest to n_1 + 1 (see Def. 9):
    n_adjusted = cryptomite.utils.closest_prime(n_1+1)
    # If n_adjusted < n_1 + 1, reduce the length of the raw
    # randomness to n_adjusted and reduce k_1 accordingly:
    if n_adjusted < n_1 + 1:
        k_1 -= n_1 + 1 - n_adjusted
        raw_randomness = raw_randomness[:(n_adjusted - 1)]
    # If n_adjusted > n_1 + 1, increase the length of
    # the raw randomness to n_adjusted by padding 0's:
    else:
        raw_randomness += [0]*(n_adjusted - n_1 - 1)
    # Take the correct number of seed bits (less than with Toeplitz):
    seed_bits = seed_bits[:n_adjusted]
    # Compute the new weak seed's min-entropy, given its
    # min-entropy rate r:
    k_2 = r * n_adjusted
    # Compute the extractor's output length (product source
    # model, see Fig. 3):
    m = floor(k_1 + k_2 - n_adjusted + 2*log2(epsilon))
    # Initialise the Circulant extractor with the appropriate parameters:
    circulant = cryptomite.Circulant(n_adjusted - 1, m)
    # Perform Circulant extraction and return the output:
    return circulant.extract(raw_randomness, seed_bits)
```

Figure 8: Improved implementation of randomness extraction for [65] using the Circulant extractor with a weak seed.

6 Conclusion and future work

We have presented `Cryptomite`, a software library of efficient randomness extractor implementations suitable for a wide range of applications. We made it easy to use and provided extensive documentation and examples to ensure that our extractors can be appended in a simple manner to any

protocols. The capacity of our extractors to tolerate large input lengths efficiently, whilst being numerically precise, makes it useful even for (semi-)device independent quantum cryptography (e.g. [25, 26, 28, 64, 65, 69–72]). We hope that our work helps to simplify the process of choosing the appropriate extractor and parameters with sufficient flexibility.

Finally, we list some future work and open questions.

- We are in the process of implementing other randomness extractors, such as an efficient (in quasi-linear computation time) version of the Raz' extractor [6] ([73]) and the ones of Hayashi et al. [7]. These may be released in subsequent versions of `Cryptomite`.
- The Trevisan extractor boasts an asymptotically small seed length. However, in practice it has the drawback that its computation time is large and, for small input lengths, the seed length is often longer than the input[13]. Some interesting future work would be making a GPU implementation of Trevisan, so that the one-bit extraction step is parallelised and fast in practice. One could alternatively find a combinatorial weak design that allows for a short seed length for small input lengths, whilst retaining minimal overlap between the chunks (see Section 4.4).
- In order to implement the Toeplitz extractor in quasi-linear computation time, the Toeplitz matrix is generated from the (weak) seed which then gets embedded into a larger circulant matrix. It would be interesting to see if there exists a deeper link between the Circulant and Toeplitz (whilst Circulant is an extension of Dodis et al.).
- To prove that some of our extractors are quantum-proof in the product source model, we employ the proof techniques of [46] and references therein which rely on obtaining a bound on the collision entropy of the extractor with a uniform seed. This bound is well understood for extractors based on two-universal hashing families. It would be interesting to find a generalised theorem that allows for the security of any two-source extractor in the quantum-proof product source model (through a bound on the collision entropy of a general extractor or otherwise).

Since the completion of this work, we have been made aware of [74], in which the authors' prove that the family of Dodis et al. extractors is quantum-proof in the Markov model with substantially better parameters than those obtained by the generic reduction of [13].

7 Acknowledgements

We thank Kevin Milner and Kieran Wilkinson for valuable feedback and comments, Sean Burton for reviewing and improving the `Cryptomite` code, Dan Neville for a useful code review of our Trevisan extractor, Sherilyn Wright for designing the repository logo and Lluis Masanes for useful discussions. We thank Martin Sandfuchs for pointing out that the generalisation to the Markov model (Theorem 4 in [13]) is unnecessary for extractors that are already quantum-proof in the product source model, enabling us to derive Corollary 1. We also acknowledge Ela Lee and Matty Hoban for testing the first version of `Cryptomite`.

[13] We remark that the seed length can be made small by reducing the output length, i.e. at the cost of the protocol's efficiency. See the exact statements of the seed length in Section 4.4

References

[1] Ronen Shaltiel. An introduction to randomness extractors. In *International Colloquium on Automata, Languages, and Programming*, pages 21–41. Springer, 2011. doi:https://doi.org/10.1007/978-3-642-22012-8_2.

[2] Gilles Van Assche. *Quantum cryptography and secret-key distillation*. Cambridge University Press, 2006. doi:https://doi.org/10.1007/978-3-642-22012-8_2.

[3] Henri Nussbaumer. *The fast Fourier transform*. Springer, 1982. doi:https://doi.org/10.1007/978-3-642-81897-4_4.

[4] Cameron Foreman, Richie Yeung, and Florian J Curchod. Statistical testing of random number generators and their improvement using randomness extraction. *Entropy*, 26(12):1053, 2024. doi:https://doi.org/10.3390/e26121053.

[5] Ariel Gabizon, Ran Raz, and Ronen Shaltiel. Deterministic extractors for bit-fixing sources by obtaining an independent seed. *SIAM Journal on Computing*, 36(4):1072–1094, 2006. doi:https://doi.org/10.1137/S009753970544704.

[6] Ran Raz. Extractors with weak random seeds. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of computing*, pages 11–20, 2005. doi:https://doi.org/10.1145/1060590.1060593.

[7] Masahito Hayashi and Toyohiro Tsurumaru. More efficient privacy amplification with less random seeds via dual universal hash function. *IEEE Transactions on Information Theory*, 62(4):2213–2232, 2016. doi:https://doi.org/10.1109/TIT.2016.2526018.

[8] Luca Trevisan. Extractors and pseudorandom generators. *Journal of the ACM*, 48(4):860–879, 2001. doi:https://dx.doi.org/10.1145/502090.502099.

[9] Eshan Chattopadhyay and Jyun-Jie Liao. Extractors for sum of two sources. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1584–1597, 2022. doi:https://doi.org/10.1145/3519935.3519963.

[10] Xin Li. Two source extractors for asymptotically optimal entropy, and (many) more. *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1271–1281, 2023. doi:https://doi.org/10.1109/FOCS57990.2023.00075.

[11] Dmitry Gavinsky, Julia Kempe, Iordanis Kerenidis, Ran Raz, and Ronald De Wolf. Exponential separations for one-way quantum communication complexity, with applications to cryptography. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, pages 516–525, 2007. doi:https://doi.org/10.1145/1250790.1250866.

[12] Marshall Ball, Oded Goldreich, and Tal Malkin. Randomness extraction from somewhat dependent sources. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022. doi:https://doi.org/10.4230/LIPIcs.ITCS.2022.12.

[13] Rotem Arnon-Friedman, Christopher Portmann, and Volkher B. Scholz. Quantum-Proof Multi-Source Randomness Extractors in the Markov Model. In *11th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2016)*, volume 61 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:34, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-019-4. doi:https://doi.org/10.4230/LIPIcs.TQC.2016.2.

[14] Yevgeniy Dodis, Vinod Vaikuntanathan, and Daniel Wichs. Extracting randomness from extractor-dependent sources. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*, pages 313–342. Springer, 2020. doi:https://doi.org/10.1007/978-3-030-45721-1_12.

[15] Jaikumar Radhakrishnan and Amnon Ta-Shma. Bounds for dispersers, extractors, and depth-two superconcentrators. *SIAM Journal on Discrete Mathematics*, 13(1):2–24, 2000. doi:https://doi.org/10.1137/S0895480197329508.

[16] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001. doi:https://doi.org/10.1109/SFCS.2001.959888.

[17] Yevgeniy Dodis, Ariel Elbaz, Roberto Oliveira, and Ran Raz. Improved randomness extraction from two independent sources. *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 334–344, 2004. doi:https://doi.org/10.1007/978-3-540-27821-4_30.

[18] Cameron Foreman, Sherilyn Wright, Alec Edgington, Mario Berta, and Florian J Curchod. Practical randomness amplification and privatisation with implementations on quantum computers. *Quantum*, 7:969, 2023. doi:https://doi.org/10.22331/q-2023-03-30-969.

[19] Renato Renner. Security of quantum key distribution. *International Journal of Quantum Information*, 6(01):1–127, 2008. doi:https://doi.org/10.1142/S0219749908003256.

[20] Hugo Krawczyk. LFSR-based hashing and authentication. In *Annual International Cryptology Conference*, pages 129–139. Springer, 1994. doi:https://doi.org/10.1007/3-540-48658-5_15.

[21] Wolfgang Mauerer, Christopher Portmann, and Volkher B Scholz. A modular framework for randomness extraction based on Trevisan's construction. *arXiv preprint arXiv:1212.0520*, 2012.

[22] Yanbao Zhang, Lynden K Shalm, Joshua C Bienfang, Martin J Stevens, Michael D Mazurek, Sae Woo Nam, Carlos Abellán, Waldimar Amaya, Morgan W Mitchell, Honghao Fu, et al. Experimental low-latency device-independent quantum randomness. *Physical review letters*, 124 (1):010505, 2020. doi:https://doi.org/10.1103/PhysRevLett.124.010505.

[23] Artur Ekert. Quantum cryptography based on Bell's theorem. *Physical Review Letters*, 67(6): 661, 1991. doi:https://doi.org/10.1103/PhysRevLett.67.661.

[24] Antonio Acín and Lluis Masanes. Certified randomness in quantum physics. *Nature*, 540(7632): 213–219, 2016. doi:https://doi.org/10.1038/nature20119.

[25] David P Nadlinger, Peter Drmota, Bethan C Nichol, Gabriel Araneda, Dougal Main, Raghavendra Srinivas, David M Lucas, Christopher J Ballance, Kirill Ivanov, Ernest Y-Z Tan, et al. Experimental quantum key distribution certified by Bell's theorem. *Nature*, 607(7920):682–686, 2022. doi:https://doi.org/10.1038/s41586-022-04941-5.

[26] Lynden K Shalm, Yanbao Zhang, Joshua C Bienfang, Collin Schlager, Martin J Stevens, Michael D Mazurek, Carlos Abellán, Waldimar Amaya, Morgan W Mitchell, Mohammad A Alhejji, et al. Device-independent randomness expansion with entangled photons. *Nature Physics*, 17(4):452–456, 2021. doi:https://doi.org/10.1038/s41567-020-01153-4.

[27] Stefano Pironio, Antonio Acín, Serge Massar, A Boyer de La Giroday, Dzmitry N Matsukevich, Peter Maunz, Steven Olmschenk, David Hayes, Le Luo, T Andrew Manning, et al. Random numbers certified by Bell's theorem. *Nature*, 464(7291):1021–1024, 2010. doi:https://doi.org/10.1038/nature09008.

[28] Peter Bierhorst, Emanuel Knill, Scott Glancy, Yanbao Zhang, Alan Mink, Stephen Jordan, Andrea Rommal, Yi-Kai Liu, Bradley Christensen, Sae Woo Nam, et al. Experimentally generated randomness certified by the impossibility of superluminal signals. *Nature*, 556(7700):223–226, 2018. doi:https://doi.org/10.1038/s41586-018-0019-0.

[29] Mayank Kharbanda. Randomness extractors, 2020. URL https://github.com/MayankKharbanda/randomness_extractors.

[30] Michele Mancusi. libtrevisan, 2019. URL https://github.com/michelemancusi/libtrevisan.

[31] Tanvirul Islam. Toeplitz extractor, 2018. URL https://github.com/tanvirulz/toeplitz_extractor.

[32] BYUCamachoLab. ottoeplitz, 2022. URL https://github.com/BYUCamachoLab/ottoeplitz.

[33] Rok Zitko. Toeplitz, 2022. URL https://github.com/rokzitko/toeplitz.

[34] Wolfgang Mauerer. libtrevisan, 2014. URL https://github.com/wolfgangmauerer/libtrevisan.

[35] Mario Berta and Fernando Brandao. Randomness generation, 2021. URL https://github.com/aws/amazon-braket-examples/blob/main/examples/advanced_circuits_algorithms/Randomness/Randomness_Generation.ipynb.

[36] John Von Neumann. Various techniques used in connection with random digits. *John von Neumann, Collected Works*, 5:768–770, 1963.

[37] Renato Renner, Stefan Wolf, and Jurg Wullschleger. The single-serving channel capacity. In

2006 IEEE International Symposium on Information Theory, pages 1424–1427. IEEE, 2006. doi:https://doi.org/10.1109/ISIT.2006.262081.

[38] Robert König and Renato Renner. Sampling of min-entropy relative to quantum knowledge. *IEEE Transactions on Information Theory*, 57(7):4760–4787, 2011. doi:https://doi.org/10.1109/TIT.2011.2146730.

[39] Robert Konig, Renato Renner, and Christian Schaffner. The operational meaning of min-and max-entropy. *IEEE Transactions on Information theory*, 55(9):4337–4347, 2009. doi:https://doi.org/10.1109/TIT.2009.2025545.

[40] Jesse Kamp and David Zuckerman. Deterministic extractors for bit-fixing sources and exposure-resilient cryptography. *SIAM Journal on Computing*, 36(5):1231–1247, 2007. doi:https://doi.org/10.1137/S0097539705446846.

[41] Renato Renner and Robert König. Universally composable privacy amplification against quantum adversaries. In *Theory of Cryptography Conference*, pages 407–425. Springer, 2005. doi:https://doi.org/10.1007/978-3-540-30576-7_22.

[42] Marco Tomamichel, Christian Schaffner, Adam Smith, and Renato Renner. Leftover hashing against quantum side information. *IEEE Transactions on Information Theory*, 57(8):5524–5535, 2011. doi:https://doi.org/10.1109/TIT.2011.2158473.

[43] Anindya De, Christopher Portmann, Thomas Vidick, and Renato Renner. Trevisan's extractor in the presence of quantum side information. *SIAM Journal on Computing*, 41(4):915–940, 2012. doi:https://doi.org/10.1137/100813683.

[44] Johan Håstad, Russell Impagliazzo, Leonid A Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999. doi:https://doi.org/10.1137/S0097539793244708.

[45] Boaz Barak, Yevgeniy Dodis, Hugo Krawczyk, Olivier Pereira, Krzysztof Pietrzak, François-Xavier Standaert, and Yu Yu. Leftover hash lemma, revisited. In *Annual Cryptology Conference*, pages 1–20. Springer, 2011. doi:https://doi.org/10.1007/978-3-642-22792-9_1.

[46] Mario Berta and Fernando Brandao. Robust randomness generation on quantum computers, 2021. URL `https://marioberta.info/wp-content/uploads/2021/07/randomness-theory.pdf`.

[47] Patrick Hayden, Richard Jozsa, Denes Petz, and Andreas Winter. Structure of states which satisfy strong subadditivity of quantum entropy with equality. *Communications in mathematical physics*, 246:359–374, 2004. doi:https://doi.org/10.1007/s00220-004-1049-z.

[48] Max Kessler and Rotem Arnon-Friedman. Device-independent randomness amplification and privatization. *IEEE Journal on Selected Areas in Information Theory*, 1(2):568–584, 2020. doi:https://doi.org/10.1109/JSAIT.2020.3012498.

[49] Roy Kasher and Julia Kempe. Two-source extractors secure against quantum adversaries. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 656–669. Springer, 2010. doi:https://doi.org/10.1007/978-3-642-15369-3_49.

[50] Amnon Ta-Shma. Short seed extractors against quantum storage. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, pages 401–408, 2009. doi:https://doi.org/10.1145/1536414.1536470.

[51] Daniela Frauchiger, Renato Renner, and Matthias Troyer. True randomness from realistic quantum devices. *arXiv preprint arXiv:1311.4547*, 2013.

[52] Frederic Dupuis, Omar Fawzi, and Renato Renner. Entropy accumulation. *Communications in Mathematical Physics*, 379(3):867–913, 2020. doi:https://doi.org/10.1007/s00220-020-03839-5.

[53] Tony Metger, Omar Fawzi, David Sutter, and Renato Renner. Generalised entropy accumulation. *Communications in Mathematical Physics*, 405(11):261, 2024. doi:https://doi.org/10.1007/s00220-024-05121-4.

[54] Frédéric Dupuis and Omar Fawzi. Entropy accumulation with improved second-order term. *IEEE Transactions on Information Theory*, 65(11):7596–7612, 2019. doi:https://doi.org/10.1109/TIT.2019.2929564.

[55] Rotem Arnon-Friedman, Frédéric Dupuis, Omar Fawzi, Renato Renner, and Thomas Vidick.

Practical device-independent quantum cryptography via entropy accumulation. *Nature Communications*, 9(1):459, 2018. doi:https://doi.org/10.1038/s41467-017-02307-4.

[56] Yanbao Zhang, Emanuel Knill, and Peter Bierhorst. Certifying quantum randomness by probability estimation. *Physical Review A*, 98(4):040304, 2018. doi:https://doi.org/10.1103/PhysRevA.98.040304.

[57] Emanuel Knill, Yanbao Zhang, and Peter Bierhorst. Generation of quantum randomness by probability estimation with classical side information. *Physical Review Research*, 2(3):033465, 2020. doi:https://doi.org/10.1103/PhysRevResearch.2.033465.

[58] Yanbao Zhang, Honghao Fu, and Emanuel Knill. Efficient randomness certification by quantum probability estimation. *Physical Review Research*, 2(1):013016, 2020. doi:https://doi.org/10.1103/PhysRevResearch.2.013016.

[59] Marco Tomamichel. *Quantum information processing with finite resources: mathematical foundations*, volume 5. Springer, 2015.

[60] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, pages 12–24, 1989. doi:https://doi.org/10.1145/73007.73009.

[61] Tzvika Hartman and Ran Raz. On the distribution of the number of roots of polynomials and explicit weak designs. *Random Structures & Algorithms*, 23(3):235–263, 2003. doi:https://doi.org/10.1002/rsa.10095.

[62] Amonrat Prasitsupparote, Norio Konno, and Junji Shikata. Numerical and non-asymptotic analysis of Elias's and Peres's extractors with finite input sequences. *Entropy*, 20(10):729, 2018. doi:https://doi.org/10.3390/e20100729.

[63] Claude Gravel. A generalization of the Von Neumann extractor. *arXiv preprint arXiv:2101.02345*, 2021.

[64] Nitin Jain, Hou-Man Chin, Hossein Mani, Cosmo Lupo, Dino Solar Nikolic, Arne Kordts, Stefano Pirandola, Thomas Brochmann Pedersen, Matthias Kolb, Bernhard Ömer, Christoph Pacher, Tobias Gehring, and Ulrik L. Andersen. Practical continuous-variable quantum key distribution with composable security. *Nature Communications*, 13(1):1–8, 2022. doi:https://doi.org/10.1038/s41467-022-32161-y.

[65] Marco Avesani, Hamid Tebyanian, Paolo Villoresi, and Giuseppe Vallone. Semi-device-independent heterodyne-based quantum random-number generator. *Physical Review Applied*, 15(3):034034, 2021. doi:https://doi.org/10.1103/PhysRevApplied.15.034034.

[66] Christian Gabriel, Christoffer Wittmann, Denis Sych, Ruifang Dong, Wolfgang Mauerer, Ulrik L Andersen, Christoph Marquardt, and Gerd Leuchs. A generator for unique quantum random numbers based on vacuum states. *Nature Photonics*, 4(10):711–715, 2010. doi:https://doi.org/10.1038/nphoton.2010.197.

[67] Johannes Thewes, Carolin Lüders, and Marc Aßmann. Eavesdropping attack on a trusted continuous-variable quantum random-number generator. *Physical Review A*, 100(5):052318, 2019. doi:https://doi.org/10.1103/PhysRevA.100.052318.

[68] Meltem Sönmez Turan, Elaine Barker, John Kelsey, Kerry A. McKay, Mary L. Baish, and Mike Boyle. Recommendation for the entropy sources used for random bit generation. *NIST Special Publication*, 800(90B):102, 2018.

[69] Wei Zhang, Tim van Leent, Kai Redeker, Robert Garthoff, René Schwonnek, Florian Fertig, Sebastian Eppelt, Wenjamin Rosenfeld, Valerio Scarani, Charles C-W Lim, et al. A device-independent quantum key distribution system for distant users. *Nature*, 607(7920):687–691, 2022. doi:https://doi.org/10.1038/s41586-022-04891-y.

[70] Yanbao Zhang, Hsin-Pin Lo, Alan Mink, Takuya Ikuta, Toshimori Honjo, Hiroki Takesue, and William J Munro. A simple low-latency real-time certifiable quantum random number generator. *Nature Communications*, 12(1):1056, 2021. doi:https://doi.org/10.1038/s41467-021-21069-8.

[71] Wen-Zhao Liu, Ming-Han Li, Sammy Ragy, Si-Ran Zhao, Bing Bai, Yang Liu, Peter J Brown, Jun Zhang, Roger Colbeck, Jingyun Fan, et al. Device-independent randomness expansion against quantum side information. *Nature Physics*, 17(4):448–451, 2021. doi:https://doi.org/10.1038/s41567-020-01147-2.

[72] Ming-Han Li, Xingjian Zhang, Wen-Zhao Liu, Si-Ran Zhao, Bing Bai, Yang Liu, Qi Zhao, Yuxiang Peng, Jun Zhang, Yanbao Zhang, et al. Experimental realization of device-independent quantum randomness expansion. *Physical Review Letters*, 126(5):050503, 2021. doi:https://doi.org/10.1103/PhysRevLett.126.050503.

[73] Cameron Foreman, Lewis Wooltorton, Kevin Milner, and Florian J Curchod. An efficient construction of Raz's two-source randomness extractor with improved parameters. *In preparation*, 2024.

[74] Jakob Miller, Martin Sandfuchs, Carla Ferradini, and Ramona Wolf. Private communication. *In preparation*, 2024.

[75] Irwin Kra and Santiago R Simanca. On circulant matrices. *Notices of the AMS*, 59(3):368–377, 2012. doi:http://dx.doi.org/10.1090/noti804.

[76] Umesh Vazirani. Efficiency considerations in using semi-random sources. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 160–168, 1987.

A Appendices

A.1 Comparison of seeded extractors

Seeded extractors are used for numerous applications. Trevisan's extractor has the asymptotically shortest seed length of the seeded extractors implemented in this work. However, this is not always the case in practise. In this subsection, we provide a figure demonstrating that the asymptotic performance is not always preserved for short input lengths. We consider only those that have quantum-proof in the seeded case with near minimal entropy loss (i.e. not Dodis et al.).

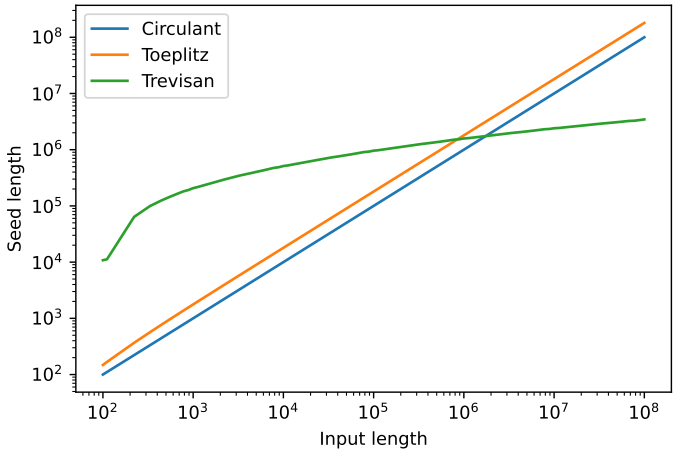

Figure 9: A plot displaying the seed length of our extractor algorithms as a function of the input length (n_1) for each seeded extractor of `Cryptomite`. The seed length is calculated by fixing the input min-entropy to $k_1 = \frac{n_1}{2}$ and calculating the maximum output length according to Figure 3.

We remark that the seed length of the Trevisan extractor depends on the output length m and the extractor error ϵ. For small m, the seed will be shorter – a situation that naturally occurs in the case of $k_1 << n_1$ and may be relevant in certain experimental situations. Alternatively, the seed length can be further reduced by using a different configuration for the *weak design*, at the cost of additional entropy loss. For a more detailed explanation, see Section 4.4.

A.2 The number-theoretic transform and the convolution theorem

The Circulant, Dodis et al. and Toeplitz extractors in `Cryptomite` can be expressed as convolutions, allowing an efficient implementation by using the fast Fourier transform (FFT) or number-theoretic Transform (NTT). However, the FFT uses floating point arithmetic, which may cause loss of precision

that is unacceptable, for example, in cryptographic applications. Instead, we use the NTT with modular arithmetic to implement the same ring operations without precision issues.

Definition 10 (Number-theoretic transform (NTT))**.** The number-theoretic transform on a vector v of length n is a discrete Fourier transform computed using the ring of integers modulo p ($\mathbb{Z}/p\mathbb{Z}$) instead of the ring of complex numbers ($\mathbb{C}$) with primitive root ω (instead of $e^{-i\frac{2\pi}{n}}$) such that $\omega^n = 1$. The inverse NTT, NTT^{-1}, is defined such that $\mathsf{NTT}^{-1}(\mathsf{NTT}(v)) = v$ and

$$\mathsf{NTT}(v)_i = \sum_{j=0}^{n-1} v_j \cdot \omega^{jk} \qquad \mathsf{NTT}^{-1}(v)_i = \sum_{j=0}^{n-1} v_j \cdot \omega^{-jk} \qquad \text{for } i = 0, \ldots, n-1. \tag{45}$$

Theorem 14 (Convolution theorem)**.** The NTT of the circular convolution of two vectors v and w of length n is the element wise product of the NTTs of v and w:

$$\mathsf{NTT}(v * w)_i = \mathsf{NTT}(v)_i \cdot \mathsf{NTT}(w)_i \qquad \text{for } i = 0, \ldots, n-1 \tag{46}$$

Proof. We modify the usual proof of the discrete convolution theorem to use $\mathbb{Z}/p\mathbb{Z}$ instead of $\mathbb{C}$.

$$\begin{aligned}
\mathsf{NTT}(v * w)_i &= \sum_{k}^{n-1} \left(\sum_{j=0}^{n-1} v_j \cdot w_{k-j} \right) \omega^{ik} && (47)\\
&= \sum_{j=0}^{n-1} v_j \omega^{ij} \sum_{k=0}^{n-1} w_{k-j} \omega^{i(k-j)} && (48)\\
&= \left(\sum_{j=0}^{n-1} v_j \omega^{ij} \right) \left(\sum_{t=0}^{n-1} w_t \omega^{it} \right) \qquad \text{Let } t = k - j && (49)\\
&= \mathsf{NTT}(v)_i \cdot \mathsf{NTT}(w)_i && (50)
\end{aligned}$$

□

The convolution theorem allows for the implementation of convolution in $O(n \log n)$ time using the NTT (or FFT), rather than the naive method requiring $O(n^2)$. This efficiency improvement makes randomness extraction feasible for input lengths $n > 10^6$, as shown in Section 2.3 "Performance".

A.3 Proofs for the Circulant extractor

In order to prove that our Circulant construction is an extractor with the parameters specified in Section 4, we must first prove that the family of functions, constructed from circulant matrices, is a two-universal family of hash functions. Once this is shown, we can exploit the well-known Leftover Hashing Lemma [60] and use Corollary 5.6.1 from [19] to prove the construction is directly quantum-proof. One of the difficulties in proving that the hash functions from circulant matrices are two-universal arises from a set of 'problematic' inputs (concretely, any pair of inputs that differ in every position), which we avoid by padding the extractor's input. This can be achieved with a single constant bit padding, in turn minimising the required amount of additional seed. We start by stating some useful Lemmas and Theorems.

Theorem 15. For any prime n and $x' \in \{0,1\}^n$, such that $x' \neq \{0\}^n, \{1\}^n$, the circulant matrix generated by x', $\mathrm{circ}(x')$, is non-singular.

Proof. The proof follows directly from considering a special case of Proposition 23 in [75], where elements of x' are bits, and noting that $x' \neq \{0\}^n, \{1\}^n$, by definition. □

Lemma 2. Any square non-singular matrix is a bijection.

Proof. This property follows directly from the invertibility of non-singular matrices. □

Lemma 3. Let a random variable $Y \in \{0,1\}^n$ be such that $p_Y(y) = 2^{-n}$ for all y. Then, for any non-singular $n \times n$ matrix D, $p_{DY}(y) = 2^{-n}$ for all y.

Proof. For each $y \in \{0,1\}^n$, there is only one value $Y = y^*$ s.t. $Dy^* = y$, and this value is unique from the fact that D is a bijection. This means that, if

$$p_Y(y) = 2^{-n} \qquad \forall y, \tag{51}$$

then

$$p_{DY}(y) = p_Y(y^*) = 2^{-n} \qquad \forall y \tag{52}$$

□

Lemma 4. Given a random variable $Y \in \{0,1\}^n$ such that $p_Y(y) = 2^{-n}$ for all $y \in \{0,1\}^n$, the first m bits of Y, denoted $Y_{0:m-1}$ satisfy

$$p_{Y_{0:m-1}}(u) = 2^{-m} \tag{53}$$

for all $u \in \{0,1\}^m$.

Proof. We use that $p_{Y_{0:m-1}}(u)$ are marginals:

$$p_{Y_{0:m-1}}(u) = \sum_{v \in \{0,1\}^{n-m}} p_{Y_{0:m-1}, Y_{m:n-1}}(u, v) \tag{54}$$

$$= \sum_{v \in \{0,1\}^{n-m}} p_Y([u,v]) = 2^{-n} 2^{n-m} = 2^{-m} \tag{55}$$

□

Theorem 16. The function in Definition 9 is a two-universal family of hash functions.

Proof. The extractor function (Circulant) in Definition 9 is a two-universal family of hash functions if, for all $x \neq \tilde{x} \in \{0,1\}^{n-1}$ and $Y \in \{0,1\}^n$ such that $p_Y(y) = 2^{-n}$ for all y, we have

$$P\left(\mathsf{Circulant}(x, Y) = \mathsf{Circulant}(\tilde{x}, Y)\right) \leq 2^{-m}. \tag{56}$$

Let $x' = [x, 0]$, $\tilde{x}' = [\tilde{x}, 0]$ and define $d' = x' \oplus \tilde{x}'$, where $\oplus$ denotes element wise addition modulo 2. Then, we can re-write the probability as

$$P\left(\mathsf{Circulant}(x, Y) = \mathsf{Circulant}(\tilde{x}, Y)\right) = P((\mathrm{circ}(d')Y)_{0:m-1} = \{0\}^m) = p_{(\mathrm{circ}(d')Y)_{0:m-1}}(\{0\}^m). \tag{57}$$

Next, $x, \tilde{x}$ differ by at least one bit (by definition), so $d' \neq \{0\}^n$ and $x', \tilde{x}'$ share the same last bit (from the padding), so $d' \neq \{1\}^n$. Then, by Theorem 15, $\mathrm{circ}(d')$ is non-singular, and since Y is uniformly distributed, we can use Lemma 3 to bound the probability $p_{\mathrm{circ}(d')Y}(y)$ for all $y \in \{0,1\}^n$. Specifically, this leads to

$$p_{\mathrm{circ}(d')Y}(\{0\}^n) = 2^{-n} \tag{58}$$

which, in combination with Lemma 4, gives

$$p_{(\mathrm{circ}(d')Y)_{0:m-1}}(\{0\}^m) = 2^{-m} \tag{59}$$

and completes our proof. □

Now, we have the ingredients to prove the main theorem, Theorem 9, which follows directly from Theorem 16 and corollary 5.6.1 from [19].

A.4 Dodis et al. (and Circulant) implementation

Concretely, our implementation of the Dodis et al. extractor follows [18] with $\{A_i\}_i$ chosen to be right cyclic shift matrices, which means losing one output bit compared the optimal choice for the extractor. We also require that n, the input length, is prime with primitive root 2 and $X = x \neq \{0\}^n, \{1\}^n$. The set of matrices $A_0, A_1 \dots A_{n-1}$ are defined as the $n \times n$ matrices

$$A_0 = \begin{bmatrix} 1 & 0 & 0 & \dots & \dots & 0 \\ 0 & 1 & 0 & \dots & \dots & 0 \\ 0 & 0 & 1 & \dots & \dots & 0 \\ \vdots & & \ddots & & \vdots & \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}, \quad A_1 = \begin{bmatrix} 0 & 1 & 0 & \dots & \dots & 0 \\ 0 & 0 & 1 & \dots & \dots & 0 \\ 0 & 0 & 0 & \dots & \dots & 0 \\ \vdots & & \ddots & & \vdots & \\ 0 & 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & 0 & \dots & 0 & 0 \end{bmatrix}, \dots, \tag{60}$$

and satisfy the constraint that the sum of any subset $B \subset \{A_i\}_i$ has rank at least $n-1$, as proved in [76]. In this form, the extractor can be re-written as the matrix-vector multiplication

$$\text{Dodis et al.}(x, y) = \left(\begin{bmatrix} x_0 & x_1 & x_2 & \dots & \dots & x_{n-1} \\ x_{n-1} & x_0 & x_1 & \dots & \dots & x_{n-2} \\ x_{n-2} & x_{n-1} & x_0 & \dots & \dots & x_{n-3} \\ \vdots & & \ddots & \ddots & & \vdots \\ x_2 & x_3 & x_4 & \dots & x_0 & x_1 \\ x_1 & x_2 & x_3 & \dots & x_{n-1} & x_0 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ \vdots \\ y_{n-1} \end{bmatrix} \right)_{0:m-1}, \tag{61}$$

where $x = x_0, x_1, \dots x_{n-1}$, $y = y_0, y_1, \dots y_{n-1}$ and the subscript $0 : m-1$ denotes the first m elements (bits) of the matrix-vector multiplication.

Although this construction is not optimal in the sense that it loses one output bit compared to the optimal construction, it allows the overall function to be re-written as a circular convolution ([18], Appendix D.1, Definition 10). This means that we can write

$$\text{Dodis et al.}(x, y)_i = \Big(\mathsf{R}(x) * y\Big)_i \tag{62}$$

$$= \sum_{j=0}^{n-1} \mathsf{R}(x)_{i-j} \cdot y_j, \tag{63}$$

where subscript $i \in \{0, \dots m-1\}$ denotes the ith output, $\mathsf{R}(x) = x_0, x_{n-1}, x_{n-2}, \dots, x_1$, and indices are mod n. The output consists of the first m elements of this convolution. The extractor algorithm is given by the pseudo-code in Algorithm 1.

Algorithm 1 Implementation of the Dodis et al. Extractor

```
function ExtDodis_{m,n}(u, v)
    L ← 2^{floor(log_2(2n−2)+1)}                ▷ Calculate NTT size
    create u′ of size L
    create v′ of size L
    for i = 0 ... n − 1 do
        u′[i] ← u[(n − i) mod n]
        v′[i] ← v[i]
    end for
    w ← conv(u′, v)
    return first m bits of w
end function
```

Total Computation Time: The Dodis et al. extractor requires two pre-processing steps of $O(n)$: first, to reduce n so that it is a prime with 2 as a primitive root, and second, to check that the

input X is not the all 0 or all 1 string. The extractor itself can be implemented using the NTT, as shown in Algorithm 1. As shown in Theorem 14, this has computation time $O(n \log n)$. Therefore, the total computation time for implementing the Dodis et al. extractor is $O(n \log n)$ (sometimes called near-linear or quasi-linear).

Circulant extractor implementation – The Circulant extractor is implemented using the Dodis et al. implementation above, with two small differences that do not affect the overall computation time. First, to check that the input length plus one (i.e. the length of x plus one) is a prime, instead of prime with primitive root 2 (which is lifted by a change in proof techniques) and second, to pad the x input string with a single (0) bit.

A.5 Toeplitz implementation

Although the definition of the Toeplitz extractor is essentially a circular convolution, the $O(n \log n)$ algorithm in Appendix A.2 expects the input and output lengths of the convolution to be equal, so it cannot be directly applied. Therefore, to implement Toeplitz in $O(n \log(n))$ time, we must embed the Toeplitz matrix from Equation (33), with m rows and n columns, within a square circulant matrix. Concretely, the Toeplitz matrix is embedded in the top-left quadrant of the square $(n+m-1) \times (n+m-1)$ circulant matrix Equation (64):

$$\operatorname{circ}(y_0, y_{n+m-2}, y_{n+m-3}, \ldots, y_1) = \begin{bmatrix} \operatorname{toep}(y) & \cdot \\ \cdot & \cdot \end{bmatrix}, \tag{64}$$

where $\operatorname{circ}(\cdot)$ denotes the circulant matrix generating function as in Equation (24) and $\operatorname{toep}(y)$ denotes the Toeplitz matrix from Equation (33). Then, by padding the weak input $x \in \{0,1\}^n$ with $m-1$ zeros (i.e., $x \to x \circ \{0\}^{m-1}$), the Toeplitz extractor can be recovered by the matrix-vector multiplication $(\operatorname{circ}(y_0, y_{n+m-2}, y_{n+m-3}, \ldots, y_1)x')_{0:m-1}$ where $x' = x \circ \{0\}^{m-1}$. Using this embedding, the Toeplitz extractor can be implemented using the NTT in $O(n \log n)$ computation time. The full implementation of the Toeplitz extractor can be seen in the pseudo-code in Algorithm 2.

Algorithm 2 Implementation of the Toeplitz Extractor

```
function ExtToeplitz_{n,m}(u, v)
    L ← 2^{floor(log_2(2n)+1)}                ▷ Calculate NTT size
    create u' of size L
    create v' of size L
    for i = 0 ... m − 1 do
        v'[i] ← v[i]
    end for
    for i = 0 ... n − 1 do
        u'[i] ← u[i]
        v'[L − n + 1 + i] = v[m + i]
    end for
    w ← conv(u', v')
    return first m bits of w
end function
```

Total Computation Time: The Toeplitz extractor requires no pre-processing steps and can be computed by the convolution implemented using the NTT, embedding the Toeplitz matrix in a larger circulant matrix. This has total computation time $O(n \log n)$.

A.6 Trevisan implementation

We follow the implementations by Mauerer et al. [21], which combines multiple iterations of the weak design from Hartman and Raz [61] to create a *block weak design*, then uses the strong Reed-Solomon

Hadamard (denoted RSH) polynomial hashing based one-bit extractor constructed from a combination of the Reed-Solomon and the Hadamard code (see [21], 'Section III. DERIVATIONS'). Informally, our implementation of the Trevisan extractor takes an input $x \in \{0,1\}^n$ and a seed $y \in \{0,1\}^d$ and extracts m output bits by

1. Using the block weak design, generate m sub-strings of length t from y that have maximum overlap $r = 1$ (where r is defined in Equation (67)).

2. Using each of these m sub-string to (respectively) seed a RSH one-bit one-bit extractor, process the extractor input x into a near-perfect bit as output.

3. Combine the m single output bits from the individual RSH one-bit extractors.

In order for the sub-string generation and one-bit extraction procedures to fit together, the implementation requires the extractor seed, y, to be of length $d = at$ (following the notation of [21]), where

$$t \in \mathbb{P}_{\geq q} \quad : \quad q = 2\lceil \log_2(n) + 2\log_2(2m/\epsilon) \rceil, \tag{65}$$

$$a = \max\{1, \left\lceil \frac{\log_2(m - 2\exp(1)) - \log_2(t - 2\exp(1))}{\log_2(2\exp(1)) - \log_2(2\exp(1) - 1)} \right\rceil\}, \tag{66}$$

where $\mathbb{P}_{\geq q}$ denotes the set of primes that are larger or equal to q and $\exp(1) \approx 2.718$ is the basis of the natural logarithm. For what follows in this section, we use ϵ_1 to denote the per-bit error (coming from the one-bit extractor) and the total error is given by $\epsilon = m\epsilon_1$. We now formalise this description.

Definition 11 (Weak design, from [21])**.** A family of sets $S_1, \ldots, S_m$, where $S_i \subset \{1, \ldots, d\}$ for all $i \in \{1, \ldots, m\}$ constitutes a (m, t, r, d)-weak design if

- Each set S_i has the same length, t.
- All sets in the family $S_1, \ldots, S_m$ have maximum overlap r, i.e. satisfies the condition

$$\forall i : \quad \sum_{j=1}^{i-1} 2^{|S_j \cap S_i|} \leq rm. \tag{67}$$

Each set S_i can be understood as a set of t indices/positions of a d length bit string. For some given bit string y, we use y_{S_i} to denote the sub-string of y, of length t, where the elements of y_{S_i} are the elements of y at indices S_i.

Definition 12 (1-bit extractor)**.** A 1-bit extractor is a strong seeded extractor $\mathsf{Ext}_s^1 : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}$, i.e. a strong seeded extractor with output length exactly 1.

The ith output bit of the Trevisan extractor, $\mathsf{Trevisan}(x, y) : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, can be written as:

$$\mathsf{Trevisan}(x, y)_i = \mathsf{Ext}_s^1(x, y_{S_i}) \tag{68}$$

where $i \in \{1, \ldots, m\}$ denotes the ith output bit, $S_1, \ldots, S_m$ are the outputs of the weak design.

Block weak design: In our implementation, we follow that of [21] and use a (Equation (66)) instances of the weak design from Hartman and Raz [61] to form a single *block weak design*. Using the Hartman and Raz weak design obliges t to be prime. The full details and parameters can be found directly in [21], 'Section III. B. Weak Designs'.

RSH 1-bit extraction: After using the block weak design to split the seed into m sub-strings, we perform m RSH one-bit extractions, where the one bit extractor is constructed from the concatenation of a Reed-Solomon and a Hadamard code (see [21], 'Section III. C. 3. Polynomial hashing'). Each

one-bit extractor takes as input the extractor input $x \in \{0,1\}^n$ and one of the m sub-strings of the seed y generated by the block weak design, $y_{S_i} \in \{0,1\}^t$ for some $i \in \{1,\ldots,m\}$. To implement RSH, the seed sub-string y_{S_i} is further split into two sub-strings of l bits, where the first is used in the Reed-Solomon step and the second in the Hadamard step. The initial RSH seed is of length $t \geq 2l$, where l is

$$l = \lceil \log_2(n) + 2\log_2(2/\epsilon_1) \rceil, \tag{69}$$

from [21], 'Section III. C. 3. Polynomial hashing'. The first step in the RSH extractor is the Reed-Solomon step. This is implemented as follows:

- Split the extractor input x into $s = \lceil n/l \rceil$ chunks x_i. Pad the last chunk x_s with 0s if need be to make all chunks of length l.
- View each chunk x_i as an individual element in the field of $\mathbb{GF}(2^l)$ i.e. the binary representation of it, for example, if $l = 3$ and $x_0 = \{0,0,1\}$ then $x_0 = 1$ in the field $\mathbb{GF}(2^3 = 8)$.
- Then, we evaluate the polynomial Equation (70) (calculated using finite field multiplication) for each of the s elements x_i in $\mathbb{GF}(2^l)$ constructed form the extractor input, where $\alpha_1 \in \mathbb{GF}(2^l)$ is the element generated by using the first l seed bits (i.e. the first half) to fix each of the l coefficients.

$$p_{\alpha_1}(x) = \sum_{i=1}^{s} x_i \alpha_1^{s-i}. \tag{70}$$

Next, we combine the Reed-Solomon step with a Hadamard Step:

- Let $\alpha_2 \in \mathbb{GF}(2^l)$ be the finite field element generated using the second l seed bits (i.e. the second half) to fix each of the l coefficients.
- Output $z = \oplus_{i=1}^{l} (\alpha_2)_i p_{\alpha_1}(x)_i$, where $\oplus$ denotes addition modulo 2.

Total Trevisan computation time: The combined computation time is

$$O(\underbrace{m\log(m)t\log(t)\log(\log(t))}_{\text{weak design}} + \underbrace{msl\log(l)}_{m \text{ 1-bit extractions}}) \tag{71}$$

and we use the following relationships to determine the overall computation time in terms of n and ϵ_1:

$$n \approx m \quad s = \left\lceil \frac{n}{l} \right\rceil \quad t \approx O\left(\log(n) + \log\left(\frac{2}{\epsilon_1}\right)\right) = O\left(\log\left(\frac{n}{\epsilon_1}\right)\right) \quad l = \frac{t}{2} = O\left(\log\left(\frac{n}{\epsilon_1}\right)\right). \tag{72}$$

This allows us to re-write Equation (71) as

$$O\left(n\log(n)\log\left(\frac{n}{\epsilon_1}\right)\log\log\left(\frac{n}{\epsilon_1}\right)\log\log\log\left(\frac{n}{\epsilon_1}\right) + n^2\log\log\left(\frac{n}{\epsilon_1}\right)\right) \tag{73}$$

$$\approx O\left(\underbrace{n\log(n)\log\left(\frac{n}{\epsilon_1}\right)}_{\text{weak design}} + \underbrace{n^2\log\log\left(\frac{n}{\epsilon_1}\right)}_{m \text{ 1-bit extractions}}\right) \tag{74}$$

The weak design could be pre-computed to reduce the overall computation time of the Trevisan extraction, but the dominant term in Equation (74) comes from the m 1-bit extractions.

A.7 Von Neumann implementation

The Von Neumann extractor is implemented in the following pseudo-code.

Algorithm 3 Implementation of the Von Neumann Extractor

```
function ExtVonNeumann(b)
out = []
    for i = 0 ... ⌊len(b)/2⌋ − 1 do
        if b[2i] ≠ b[2i + 1] then
            out.append(b[2i])
        end if
    end for
    error                                   ▷ Error if all bits are the same.
end function
```

Total Computation Time: Trivially, this extractor can be implemented with linear computation time ($O(n)$).