IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2025, No. 4, pp. 145–171. DOI:10.46586/tches.v2025.i4.145-171

Entropy extractor based high-throughput post-processings for True Random Number Generators

Yifan Dang[1,4], Miloš Grujić[2], Bohan Yang[1,4,*], Wenping Zhu[1,4], Hanning Wang[1,4], Min Zhu[3], Ingrid Verbauwhede[2] and Leibo Liu[1,4,*]

[1] Beijing National Research Center for Information Science and Technology, School of Integrated Circuits, Tsinghua University.

[2] COSIC, KU Leuven, Leuven, Belgium.

[3] Wuxi Micro Innovation Integrated Circuit Design Co., Ltd., Jiangsu Wuxi, China.

[4] State Key Laboratory of Cryptography and Digital Economy Security, Tsinghua University, Beijing, 100084, China.

dangyf24@mails.tsinghua.edu.cn, {bohanyang,zhuwp,wanghn,liulb}@tsinghua.edu.cn; {milos.grujic,ingrid.verbauwhede}@esat.kuleuven.be, zhumin@mucse.com

[*] Corresponding Author.

Abstract. In cryptographic systems, true random number generation is essential, as a compromised TRNG could lead to a security catastrophe. The raw random numbers are discrete values that are derived at discrete points in time from a noise source of a TRNG. These values often exhibit statistical defects that require post-processing, also called conditioner, to improve uniformity. The two main types of post-processing are algorithmic post-processing and cryptographic post-processing, both of which have pros and cons in theories and applications. However, another type of post-processing existing between these two types, named entropy extractor, has often been overlooked by the applied cryptographic community. Therefore, we implement two information-theoretically provable entropy extractors: Toeplitz extractor and Trevisan extractor catering to various performance requirements and applications of high-throughput TRNG post-processing. This paper proposes a combination of matrix chunking and FFT acceleration to boost the performance of the Toeplitz extractor, along with a modified Toeplitz matrix design to decrease the hardware consumption. In addition, we introduce a lightweight single-bit extractor to implement an efficient Trevisan extractor. Both algorithms are devised and verified through FPGA hardware simulations. The enhanced Toeplitz extractor achieves a throughput of 42 Gbps, while the Trevisan extractor attains 1.82 Gbps, representing an 84% and 73% improvement in throughput-to-area ratio over the previous best-performing design for each extractor. The standard statistical test suites, such as NIST SP800-22, NIST SP800-90B, and AIS-31, are adopted to evaluate the effectiveness of the proposed post-processing techniques. Naturally, this approach can only serve as a supplementary measure, as modern standards, such as AIS-31, necessitate formal analysis and stochastic models to account for randomness.

Keywords: True random number · post-processing · entropy extractor · hardware security

Licensed under Creative Commons License CC-BY 4.0.
Received: 2025-04-15 Accepted: 2025-06-15 Published: 2025-09-05

1 Introduction

1.1 Application and Motivation

True random numbers are widely used across various industrial and scientific fields, particularly in cryptography. Almost all existing encryption algorithms rely on true random numbers that exhibit uniformity and unpredictability to generate encryption keys [RYDV16]. Additionally, true random numbers are used to generate initialization vectors and masks. However, most current True Random Number Generators (TRNGs) cannot generate ideal random bits directly without post-processing [ZNLZ16]. The post-processing module is a deterministic function that extracts and densifies randomness from raw random numbers to enhance their statistical properties. A functional and efficient post-processing module is crucial in the generic architecture of a TRNG, as shown in Figure 1. The entropy extractor, also referred to as conditioner in NIST SP800-90B [TBK+18], is one of the most significant means of implementing the post-processing module. Entropy extractors aim to extract near-perfect randomness from raw data generated by a practical TRNG, often aided by a random seed from another independent randomness source. While there is extensive research on entropy sources [YRG+18, FBDS23, SYR+23, DVK14], studies on the post-processing are relatively limited, with most focusing on theoretical aspects rather than practical hardware implementations [GGBB21, LSZZ23].

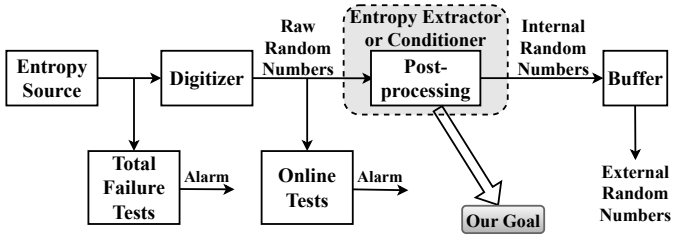

Figure 1: The generic architecture of a TRNG.

A 10-Gbit/s system is already in use in classical optical communication [TDL+06] and requires approximately a 40 Gbps random bit stream [NHL+15]. Furthermore, data throughput in current communication centers is expected to increase. Unfortunately, the throughputs of AIS-20/31 compliant TRNG cores fall short of these requirements [PMB+16], with the highest throughput being only 245Mbps, as shown in Table 1. Even an ultra-fast QRNG with a throughput of 21 Gbps cannot meet this demand [ZBY+22]. Therefore, a post-processing module is necessary to extract uniformity from multiple parallel entropy sources with dependencies to satisfy the required throughput. However, current post-processing algorithms are designed to extract randomness from either a single source or from multiple independent sources. As a result, our focus is on developing entropy extractors capable of handling multiple correlated entropy sources and generating almost uniform random numbers. Two previously proposed entropy extractors, the Toeplitz extractor [GL89] and the Trevisan extractor [RRV02], can satisfy this requirement. Both have been verified to be information-theoretically sound and are commonly used in Quantum Random Number Generators (QRNGs). These two entropy extractors offer high-security features while allowing for simple hardware implementations.

This work aims to provide a guide to entropy extractors for TRNGs in various high-speed application scenarios. For servers and data centers that handle large volumes of data, post-processing modules are essential to process raw random numbers from multiple correlated entropy sources. In these applications, throughput and extraction efficiency are the primary concerns, while the hardware area is of secondary importance. In contrast, lightweight cryptographic applications require high-throughput true random numbers

within a compact hardware resource. To address these two distinct application scenarios, we utilize two entropy extractors, the Toeplitz extractor and the Trevisan extractor, to achieve effective post-processing.

Table 1: Throughput results of the AIS-20/31 compliant TRNG cores.

TRNG type \ FPGA device	Spartan 6	Cyclone V	SmartFusion 2
ERO	0.0042 Mbps	0.0027 Mbps	0.014 Mbps
COSO	0.54 Mbps	1.44 Mbps	0.328 Mbps
MURO	2.57 Mbps	2.2 Mbps	3.62 Mbps
PLL	0.44 Mbps	0.6 Mbps	0.37 Mbps
TERO	0.625 Mbps	1 Mbps	1 Mbps
STR	154 Mbps	245 Mbps	188 Mbps

1.2 Relative works and our contributions

The Toeplitz extractor operates by multiplying the original input sequence with a binary Toeplitz matrix to produce the output sequence. The vector used to construct the binary Toeplitz matrix is referred to as the random seed. A block-parallel algorithm proposed in 2012 [HfWCl+12] performed randomness extraction on Field Programmable Gate Array (FPGAs) for the first time. A Toeplitz hash algorithm based on Fast Fourier Transformation (FFT) acceleration presented in 2018 achieved 1 Gbps throughput on Graphics Processing Units (GPUs) [WZYG18]. In 2021, Luo increased the throughput to 6.8 Gbps with pipeline computational units and FFT acceleration on GPUs [LLY+21]. However, post-processing speed is still a bottleneck for QRNG, which can achieve 21 Gbps throughput [ZBY+22]. Therefore, we need a solution to improve the processing speed further without increasing the hardware consumption.

The Trevisan extractor consists of single-bit extractors and a weak design. In addition, its seed length is logarithmic to the input length. However, the computational process of the Trevisan extractor is more complex, leading to a lower output rate. Despite its theoretical advantages, hardware implementations of the Trevisan extractor remain scarce. The output rates of some hardware implementations are relatively low, such as 0.7 kbps [MXX+13] or 20 Mbps [MPS12], which are far from meeting the demand for high-throughput true random numbers.

This study aims to improve the throughput and reduce the resource consumption of these two post-processing algorithms through parameter space exploration and hardware optimization while ensuring that the rate requirements of various high-speed application scenarios are met. The contributions of this paper are the following:

1. We are the first to establish a guide to entropy extractor based high-throughput post-processings for TRNGs and to develop both the Toeplitz and Trevisan extractors tailored for distinct application scenarios. For ASIC platforms, we pioneer the implementations of both the Toeplitz extractor and the Trevisan extractor, while for FPGA platforms, we unveil the deployment of the Trevisan extractor. We also release the source code[1] for extensive analysis of entropy extractor implementations.

2. We devise a streamlined single-bit extractor for constructing the Trevisan extractor. This single-bit extractor employs a one-time indexing technique to achieve entropy-stabilized sampling. Additionally, it integrates a strong blender based on cyclic shift matrices to hash the sampling sequence. These advancements bring about a 73% improvement in the throughput-to-area ratio and loosen the seed length requirement

[1] https://github.com/ybhphoenix/TRNG_Toeplitz_and_Trevisan_post-processing

by reducing it from $l \cdot \log_2(n)$ to $l + \log_2(n)$, where l is the sample length and n is the input length.

3. We propose an architecture for the Toeplitz extractor that synthesizes the matrix chunking and FFT acceleration optimizations, incorporating a modified Toeplitz matrix. This revised matrix halves the seed length and boosts throughput to 4 times the original. This design delivers a six-fold performance enhancement over prior Toeplitz extractors and an 84% augmentation in efficiency compared to the state of the art in cryptographic post-processings.

This paper is organized as follows. In Section 2, we introduce the background and notation used in the paper. Section 3 presents the principle of the two optimized post-processing algorithms. The hardware implementation of post-processing circuits is given in Section 4. Section 5 provides the test results of the post-processing circuits. Section 6 concludes the paper.

2 Background

2.1 Basic Notation

- X – Random variables.
- x_i – Possible outcome of a random variable X.
- $X \leftarrow \mathcal{X}$ – Variable X is sampled from distribution $\mathcal{X}$.
- $Pr(a)$ – The probability of the event a.
- U_t – A random variable that is uniform over $\{0,1\}^t$.
- $[t]$ – $\{1, 2, ..., t\}$.
- $|Z|$ – The number of elements in the set Z.
- (X, Y) – The concatenation of two random variables X and Y.
- $H_\infty(X)$ – The min-entropy of a discrete random variable X: $H_\infty(X) = -\log_2 (\max Pr(X = x_i))$.
- $\|X - Y\|_s$ – The statistical distance of the two discrete random variables X, Y taking values in the finite set I is computed as follows: $\|X - Y\|_s = \frac{1}{2} \sum_{x_i \in I} |Pr(X = x_i) - Pr(Y = x_i)|$.

2.2 Existing post-processing methods

Existing post-processing algorithms can be divided into algorithmic and cryptographic post-processing. Algorithmic post-processing provides a straightforward approach to processing raw random numbers with information-theoretical security proofs. These algorithms typically feature compact hardware implementations and low computational complexity, such as the parity filter [SV84], Von Neumann's algorithm [Neu51], resilient functions [SMS07], and Blum's algorithm [Blu86]. However, the first three methods fail to adequately eliminate the correlations in the raw bits, while Blum's algorithm cannot be applied to entropy sources that cannot be modeled by finite Markov chains. In addition, these methods are limited to processing raw random numbers from a single TRNG. In practical applications, however, the primary challenge is eliminating complex correlations from multiple dependent TRNGs. Cryptographic post-processing employs

cryptographic primitives, such as hash functions [GDS+20, TBK+18, MK23] and block ciphers [SB15], to extract randomness. These primitives are generally recommended for post-processing due to their computational security proofs [TBK+18, KG04]. However, implementing cryptographic primitives typically results in considerable energy consumption. In digital systems equipped with cryptographic accelerators, non-TRNG applications may be excluded from utilizing these accelerators during cryptographic post-processing. Moreover, cryptographic post-processing does not provide an information-theoretical security proof.

There is a category of post-processing algorithms that falls between these two types, which we refer to as "intermediate algorithms." These algorithms are more complex than algorithmic post-processing but still simpler than block ciphers and hash functions. They have a broader range of applications than the other two types of post-processing due to their ability to extract entropy from multiple correlated sources effectively.

Our focus is on post-processing algorithms that can be implemented by entropy extractors whose security proof is not based on any computational assumption or stochastic model, a feature often referred to as unconditional security. Notably, the two entropy extractors we implement, the Toeplitz extractor and the Trevisan extractor, are the representatives of these algorithms. A comparison of the three types of post-processing is presented in Table 2.

Table 2: Comparison of three types of post-processing.

Type	Item	Constant output rate	Compression rate	Handling arbitrary correlation	Low entropy loss	Simple implementation	Unconditional security
Algorithmic	XOR	✓	n	✗	✗	✓	✗
	VN	✗	≈ 4	✗	✗	✓	✗
	Blum	✗	≈ 4	✗	✗	✓	✗
Entropy extractor	Toeplitz	✓	n/m	✓	✓	✓	✓
	Trevisan	✓	n/m	✓	✗	✓	✓
Cryptographic	AES	✓	$2 \sim 3$	✓	✗	✗	✗
	SHA-2	✓	$2 \sim 5$	✓	✗	✗	✗

2.3 Entropy source models and entropy extractor

In 1986, Blum [Blu86] considered the source that could be modeled by Markov chains with finite states. Blum also introduced an entropy extraction algorithm capable of extracting nearly unbiased bits from such sources in linear time, with the extracted bits almost reaching the source's entropy. However, in practice, most entropy sources cannot be modeled by Markov chains, as pointed out by Santha and Vazirani [SV86]. They relaxed the constraints on the physical source and proposed the SV model.

Definition 1. *A distribution $\mathcal{X}$ is an SV-source if there exists a constant δ, $0 < \delta \leq \frac{1}{2}$, such that for every i and for every string s of length $i-1$, $\delta \leq Pr(X_i = 1|X_1...X_{i-1} = s) \leq 1-\delta$.*

Furthermore, Santha and Vazirani demonstrated that no deterministic entropy extraction algorithm can extract even one unbiased bit from a single SV source [SV86]. Chor and Goldreich proposed the probability-bounded sources(PRB model) based on the SV model and the current definition of the weak entropy source [CG85].

Definition 2. *A distribution $\mathcal{X}$ which outputs an infinite sequence of bits is a PRB-source ((l,b) source) if for every prefix α of the output sequence and every l bits string β, the conditional probability that the next l bits output by $\mathcal{X}$ are β is at most 2^{-b}, which means $Pr(\beta|\alpha) \leq 2^{-b}$.*

The PRB model establishes that the min-entropy of an l-bit sequence produced by the entropy source is bounded by b. Notably, an SV source with parameter δ can be considered a PRB source with parameters $l = 1$ and $b = \log_2(1 - \delta^{-1})$. Therefore, the PRB model is a more general version of the SV model, capable of describing a broader range of behaviors in physical sources. All the entropy sources discussed later in this paper are derived from the PRB model. The entropy source described within the PRB framework is referred to as a weak entropy source.

Since unbiased bits cannot be extracted from a single weak entropy source, existing extractors can generally be classified into two types: multiple-input extractors and seeded extractors [Joh18]. The multiple-input extractor requires several independent weak entropy sources, while the seeded extractor utilizes a single weak entropy source along with a section of random seed. As a key example of the multiple-input extractor, we first introduce the concept of the strong blender.

Definition 3. *Let $b_X, b_Y > 0$, $\varepsilon > 0$. A function BLE: $\{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m$ is a (b_X, b_Y, ε) strong blender(SB) if for all weak entropy source X over $\{0,1\}^n$ with $H_\infty(X) \geq b_X$ and all weak entropy source Y over $\{0,1\}^n$ with $H_\infty(Y) \geq b_Y$, we have $\|(Y, BLE(X,Y)) - (Y, U_m)\|_s \leq \varepsilon$* [DEOR04].

In practice, multiple independent entropy sources are not always available. Both the Toeplitz extractor and the Trevisan extractor are seeded extractors. This study focuses on seeded extractors (hereafter, all extractors discussed are referred to as seeded extractors), which are defined as follows:

Definition 4. *Let $b > 0$, $\varepsilon > 0$. A function Ext: $\{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (b, ε) extractor if for every weak entropy source X over $\{0,1\}^n$ with $H_\infty(X) \geq b$, we have $\|Ext(X, U_d) - U_m\|_s \leq \varepsilon$.*

The U_d mentioned above is the random seed in the seeded extractor. X is a source of random bits and the output is ε-close to uniform. In most cases, not only do we need the output to be uniform enough, but we also need the output to be independent of the seed. A more restrictive extractor named strong extractor [MXX+13] meets this demand.

Definition 5. *Let $b > 0$, $\varepsilon > 0$. A function Ext: $\{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (b, ε) strong extractor if for every weak entropy source X over $\{0,1\}^n$ with $H_\infty(X) \geq b$, we have $\|(Ext(X, U_d), U_d) - U_{m+d}\|_s \leq \varepsilon$.*

2.4 Toeplitz extractor

We begin by introducing the universal hash function, noting that the Toeplitz matrix serves as a universal hash function with proven entropy extraction capabilities [HILL99]. A universal hash function is defined in such a way that it limits the probability of different inputs mapping to the same output.

Definition 6. *F is a universal hash function family if for every function $f \in F$ and for every $v_1 \neq v_2 \in V$, we have $Pr(f(v_1) = f(v_2)) \leq \gamma$. Then any $f \in F$ is a γ-almost universal hash function. If $\gamma = \frac{1}{|Z|}$, any $f \in F$ is a universal hash function* [CW79].

A Toeplitz extractor is a seeded extractor in which the random seed is a binary Toeplitz matrix, a matrix where all the elements along the main diagonal are identical. The extraction process of this extractor is to multiply the binary Toeplitz matrix with the raw random number vector to produce the internal random number, as shown in Equation (1).

The theoretical foundation of the Toeplitz extractor is Leftover Hash Lemma [ILL89].

$$\begin{bmatrix} s_m & s_{m+1} & \cdots & s_{m+n-1} \\ \vdots & \vdots & \ddots & \vdots \\ s_2 & s_3 & \cdots & s_{n+1} \\ s_1 & s_2 & \cdots & s_n \end{bmatrix} \times \begin{bmatrix} d_1 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix} = \begin{bmatrix} a_1 \\ \vdots \\ a_{m-1} \\ a_m \end{bmatrix}. \tag{1}$$

Lemma 1. *Let $F = \{f_1, f_2, ..., f_{2^d}\}$ be a universal hash function family, mapping from $\{0,1\}^n$ to $\{0,1\}^m$, and X be a random variable over $\{0,1\}^n$ with $H_\infty(X) \geq b$. Then for $Y \in U_d$ and $f_Y \in F$, $(f_Y(X), Y)$ is $\varepsilon = 2^{(m-b)/2}$ close to U_{m+d}. That means it forms a (b, ε) strong extractor which maps $\{0,1\}^n \times \{0,1\}^d$ into $\{0,1\}^m$* [MXX+13].

The Leftover Hash Lemma shows that the Toeplitz matrix can be used as a strong extractor. The statistical distance between the extractor's output and the uniform random variable ε is called the safety factor. The Leftover Hash Lemma also bounds the min-entropy of the output sequence. Due to $m \leq b - 2\log_2(1/\varepsilon)$, the entropy loss of the Toeplitz extractor is at least $2\log_2(1/\varepsilon)$. If we decrease m, ε will also be reduced, which means that the output sequence will be closer to uniform. However, the entropy loss and the output rate will be reduced simultaneously. $\varepsilon = 2^{-80}$ is the recommended security level [Sko16], that is, $m \leq b - 160$. The dimensions of the Toeplitz matrix m and n will be greater than 10^3 and the min-entropy of the input b is greater than 10^3 in most cases. Reducing b by 160 bits does not greatly affect the entropy loss and the output rate.

2.5 Trevisan extractor

Trevisan extractor is another widely used seeded extractor and is also a strong extractor. As mentioned above, the Trevisan extractor consists of single-bit extractors and a weak design algorithm. Figure 2 illustrates the construction of the Trevisan extractor. The weak design divides the random seed into several sets with a little overlap and inserts each set into a separate single-bit extractor. The randomness in the input sequences could deal with these additional correlations, provided that the overlap is not very large [MPS12]. Nisan-Wigderson pseudo-random generator is one method that can achieve this goal [RRV02].

Traditional algorithmic post-processing methods, such as the parity filter, Von Neumann's algorithm and linear correctors, are ineffective in fully eliminating correlations present in raw random numbers. In contrast, cryptographic post-processing techniques can address these correlations, but they lack formal information-theoretical security proofs. Additionally, none of these methods are capable of processing raw random numbers derived from multiple dependent entropy sources, an important application scenario for high-speed random number generation. However, the Trevisan extractor overcomes these limitations. It can efficiently eliminate bias and correlation in raw random bits obtained from multiple dependent entropy sources, while ensuring unconditional security.

We define the whole random seed of the Trevisan extractor as $y \in \{0,1\}^d$. The random seeds for each single-bit extractor is $\{S_i\}$ such that for every index i, $|S_i| = t$ and $S_i \subset [d]$. A family of sets $S_1, ..., S_m \subset [d]$ is a weak (m, t, r, d)-design, if for every i we have

$$|S_i| = t, \sum_{j=1}^{i-1} 2^{|S_j \cap S_i|} \leq rm. \tag{2}$$

The r in Equation (2) denotes the seed overlap for each single-bit extractor. y_{S_i} is a string of length t that is formed by the bits of y at the positions given by the elements in S_i. For a given single-bit extractor $E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}$ and a set family $\{S_i\}_{i=1}^m$, the Trevisan extractor can be presented as a series connection of m extractor outputs: $Ext(x, y) = E(x, y_{S_1})...E(x, y_{S_m})$.

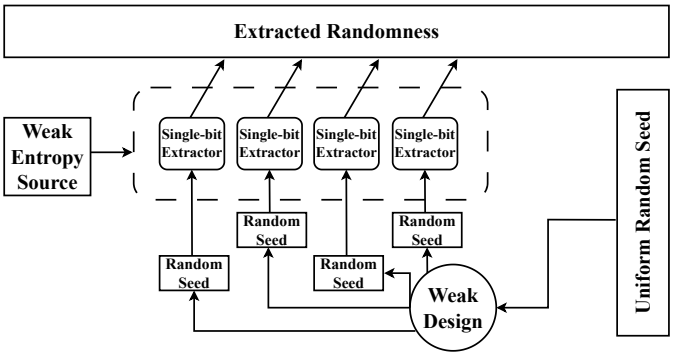

Figure 2: Construction of the Trevisan extractor.

To analyze the parameters of Trevisan extractors, we first need to introduce the concept of a classical (or quantum)-proof strong extractor. An extractor is considered a classical (or quantum)-proof strong extractor if its output is close to uniform and independent of the classical (or quantum) side information. It can be shown that any single-bit extractor is a quantum-proof strong extractor [KT08]. Therefore, a Trevisan extractor constructed from multiple single-bit extractors is also a quantum-proof strong extractor.

Theorem 1. *Let Ext:* $\{0,1\}^n \times \{0,1\}^t \to \{0,1\}$ *be a (b,*ε*) strong extractor. Then Ext is a (*$b + \log_2(1/\varepsilon)$*,*$3\sqrt{\varepsilon}$*) quantum-proof strong extractor* [MPS12].

If we have shown that *Ext* is a classical-proof strong extractor, Theorem 1 will become Lemma 2.

Lemma 2. *Let Ext:* $\{0,1\}^n \times \{0,1\}^t \to \{0,1\}$ *be a (b,*ε*) classical-proof strong extractor. Then Ext is a (b,*$(1+\sqrt{2})\sqrt{\varepsilon}$*) quantum-proof strong extractor* [MPS12].

Now, if we know the parameters of the single-bit extractor and weak design, we can calculate the parameters of the Trevisan extractor by the following Theorem [DPVR12].

Theorem 2. *Let E:* $\{0,1\}^n \times \{0,1\}^t \to \{0,1\}$ *be a (b,*ε*) quantum-proof strong extractor with uniform seed and a family of sets* $S_1, ..., S_m \subset [d]$ *a weak* (m,t,r,d)*-design. Then the Trevisan extractor,* Ext_E*:* $\{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$*, is a* $(b + rm, m\varepsilon)$ *quantum-proof strong extractor* [MPS12].

We point out that the single-bit extractor is the critical factor limiting the Trevisan extractor's performance, which is also our focus. The two dominant single-bit extractors available are XOR-code [IJK09] and polynomial hashing [MPS12].

Definition 7. *Let safety factor be* $\varepsilon > 0$*, input length* n *and the selected bits* l*. Then the XOR-code* $C : \{0,1\}^n \times [n]^l \to \{0,1\}$ *calculates the XOR value of the* l *bits selected from the input sequence:*

$$(x, i_1, ..., i_l) \mapsto \bigoplus_{j=1}^{l} x_{i_j}. \tag{3}$$

XOR-code is a lightweight single-bit extractor and a (b,ε) classical-proof strong extractor. By Lemma 2, it is also a $(b,(1+\sqrt{2})\sqrt{\varepsilon})$ quantum-proof strong extractor.The b of XOR-code satisfies the Equation (4) [MPS12].

$$b = h\left(\frac{\ln 2}{l}\log_2\left(\frac{2}{\varepsilon}\right)\right)n + 3\log_2\left(\frac{1}{\varepsilon}\right) + \log_2\left(\frac{4}{3}\right). \tag{4}$$

The h in Equation (4) is the binary Shannon entropy function. The seed length of the XOR-code is $t = l\ \log_2(n)$, where the selected bits l are correlated with the ε and the entropy loss ratio η. Their relationship satisfies $\eta = h(\frac{\ln 2}{l}\log_2(\frac{2}{\varepsilon}))$ [IJK09]. Let input length be n and input entropy density α. Then the entropy loss ratio of the XOR-code is $\eta = \alpha - 1/n$. In practice, we can obtain the value of n and α in advance and calculate the entropy loss ratio η. Let $\varepsilon' = (1+\sqrt{2})\sqrt{\varepsilon}$, XOR-code is a ($b$,$\varepsilon'$) quantum-proof strong extractor. Therefore, the seed length of the XOR-code is calculated as follows [MPS12]:

$$t = l\ \log_2(n) = \frac{2\ln 2}{h^{-1}(\alpha - 1/n)}\log_2(n)\log_2\left(\frac{2+\sqrt{2}}{\varepsilon'}\right). \tag{5}$$

Unlike the XOR-code, the polynomial hashing is another single-bit extractor with higher performance but more complex computation. Polynomial hashing performs a two-stage extraction process on the input sequence. The seed of the first extraction process is λ and the second is μ. The seed lengths of the two extraction processes are both l. Before randomness extraction, we need to partition the input sequence into s blocks $x = \{x_1, ..., x_s\}$(padding the last string x_s with 0s if necessary). The first extraction process performs polynomial multiplication and addition on the input sequence: $p_1(x) = \sum_{i=1}^{s} x_i \lambda^{s-i}$. Due to $\lambda \in GF(2^l)$, the entire polynomial computation is performed in $GF(2^l)$ and $p_1(x) \in GF(2^l)$. The second extraction process performs the inner product of $p_1(x)$ and the second seed: $p_2(x) = \bigoplus_{i=1}^{l} \mu_i p_1(x)_i$. Polynomial hashing is a ($4\log_2(1/\varepsilon) + 2$,$\varepsilon$) quantum-proof strong extractor [TSSR11] and its seed length is $t = 2l = 2\lceil \log_2(n) + 2\log_2(2/\varepsilon)\rceil$, which is independent of the input entropy density and the entropy loss ratio.

A significant limitation of polynomial hashing is its relatively slow computational speed, primarily due to the complexity of the involved calculations. The first stage involves exponential and multiplicative operations over finite fields, requiring s rounds. As a crucial factor, the degree s of the polynomial hashing grows approximately linearly with the input length n, which can become quite large in practical applications. As stated in Ref [MPS12], the need to evaluate the polynomial for every extracted bit further limits the suitability of polynomial hashing for performance-intensive scenarios.

3 Improved Algorithm

3.1 The optimization of Toeplitz extractor

3.1.1 Toplitz matrix chunking algorithm

The matrix chunking algorithm is a primary method to improve the output rate by dividing the Toeplitz matrix into smaller blocks, each of which is multiplied by the input sequence and then accumulated. In particular, we can split a $m \times n$ Toeplitz matrix into n/k sub-matrices of size $m \times k$, effectively breaking down the matrix-vector multiplication into n/k steps. In each step, a $m \times k$ sub-matrix is multiplied with an input vector of length k, producing an intermediate vector of length m. After completing all the steps, we

accumulate the intermediate vectors to obtain the final output vector.

$$\begin{aligned}&\begin{bmatrix} s_m & s_{m+1} & \cdots & s_{m+n-1} \\ \vdots & \vdots & \ddots & \vdots \\ s_2 & s_3 & \cdots & s_{n+1} \\ s_1 & s_2 & \cdots & s_n \end{bmatrix} \times \begin{bmatrix} d_1 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix} \\ &= \sum_{\substack{i=0 \\ i+=k}}^{n-k} \left(\begin{bmatrix} s_{m+i} & \cdots & s_{m+k-1+i} \\ \vdots & \ddots & \vdots \\ s_{2+i} & \cdots & s_{k+1+i} \\ s_{1+i} & \cdots & s_{k+i} \end{bmatrix} \times \begin{bmatrix} d_{i+1} \\ \vdots \\ d_{i+k-1} \\ d_{i+k} \end{bmatrix} \right) = \begin{bmatrix} a_1 \\ \vdots \\ a_{m-1} \\ a_m \end{bmatrix}\end{aligned} \tag{6}$$

3.1.2 FFT accelerated real-valued Toeplitz matrix multiplication

Another way to improve the throughput of the Toeplitz extractor is to use FFT to accelerate the matrix-vector multiplication [MXX+13]. First, we need to transform a $m \times n$ Toeplitz matrix into a $(m+n-1) \times (m+n-1)$ cyclic matrix and pad the input vector of length n with zeros to expand it into a vector of length $m+n-1$. Let C be the cyclic matrix from the Toeplitz matrix transformation and d the expanded input vector. F denotes the Fourier transformation and the multiplication is as follows:

$$K = d \times C = d \cdot F(F^{-1}CF)F^{-1} = d \cdot F \cdot diag(f_g) \cdot F^{-1} = ifft[fft(g). * fft(d)]. \tag{7}$$

The g in Equation (7) denotes the first row of the cyclic matrix C, and $diag(f_g)$ is the diagonal matrix formed by g. The multiplication of the diagonal matrix and the vector is simple multiplication by bits, which can be implemented by making bitwise AND operations. We take the first m bits from the multiplication result K as the final output. The total computational complexity of this type of matrix-vector multiplication is $O(n\ logn)$, which is lower than $O(n^2)$, the computational complexity of direct multiplication.

In practice, most of the available FFT hardware cores are designed to compute the FFT of a complex sequence, while both the input vector d and the first row of the cyclic matrix g are real sequences of length $n+m-1$. Most existing methods input d to the real-valued port and zeros to the imaginary-valued port, which wastes computing and storage resources. We can use a real-valued oriented FFT algorithm [SJHB87] to solve this problem. The relative FFT transformation results D and G can be computed using a single complex-valued FFT core instead of two. Specifically, we input the vector d to the real-valued port and the first row of the cyclic matrix g to the imaginary-valued port. We assume that $z = d + i \cdot g$ and we can obtain the transformed result $Z = FFT(z)$, which is a complex vector of length $N = n+m-1$. We use the real and imaginary parts of Z to calculate the $Re[D(k)] = \frac{1}{2}(Re[Z(k)] + Re[Z(N-k)])$ and $Re[G(k)] = \frac{1}{2}(Im[Z(k)] + Im[Z(N-k)])$. Then we perform a bitwise AND operation between $Re(D)$ and $Re(G)$ to get the intermediate vector MED. We input MED to the real-valued port of the FFT module and shift the result to the right by $\lceil \log_2(n+m-1) \rceil$ bits to achieve the IFFT operation. Figure 3 shows the whole process.

3.1.3 Toplitz matrix chunking algorithm combined with FFT acceleration

Directly using FFT to accelerate the multiplication of large matrices and vectors results in minimal throughput improvement [NHL+15]. Therefore, we divide the large Toeplitz matrix into smaller sub-matrices by rows and columns and proportionally chunk the input vector. We then apply the FFT to accelerate the multiplication of each sub-matrix with its corresponding sub-vector. The intermediate results are combined and accumulated to produce the final output. Figure 4 illustrates this process in detail.

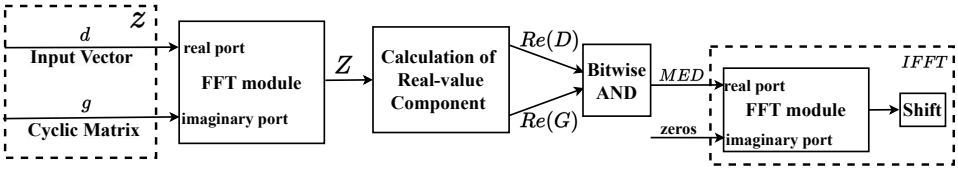

Figure 3: The whole process of the real-valued oriented FFT algorithm.

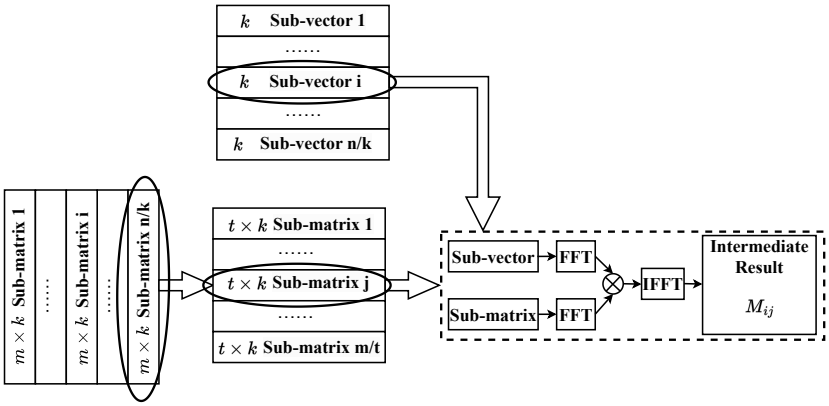

Figure 4: The combination of the Toeplitz matrix chunking algorithm and FFT acceleration.

We begin by dividing an $m \times n$ Toeplitz matrix into n/k sub-matrices with dimensions $m \times k$ by separating the columns. Simultaneously, we split the input vector into n/k sub-vectors, each of length k. Next, each $m \times k$ sub-matrix is further divided into m/t sub-matrices, each of size $t \times k$. We then compute the multiplications of these m/t $t \times k$ sub-matrices with a corresponding k-length sub-vector using an FFT accelerator. This process yields m/t intermediate sub-vectors, each of length t, within a single clock cycle. We combine these m/t intermediate sub-vectors to form an intermediate vector of length m during the same clock cycle.

As a result, we require n/k clock cycles to obtain n/k intermediate vectors. Finally, we accumulate all the intermediate vectors to produce the final output vector of length m.

3.1.4 A modified Toeplitz matrix

We can use a modified Toeplitz matrix to improve the throughput further without weakening the security [Hay11]. A $m \times n$ modified Toeplitz matrix comprises a $m \times m$ unit matrix and a $m \times (n-m)$ Toeplitz matrix.

$$G = (I_m|T) = \begin{bmatrix} 1 & & & t_m & t_{m+1} & \dots & t_{n-1} \\ & 1 & & t_{m-1} & t_m & \dots & t_{n-2} \\ & & \ddots & \vdots & \vdots & \ddots & \vdots \\ & & 1 & t_1 & t_2 & \cdots & t_{n-m} \end{bmatrix}. \tag{8}$$

Let X_n be the input vector and Y_m be the output vector. The computational process

using a modified Toeplitz matrix is shown as follows:

$$\begin{aligned} Y_m = G_{m\times n} \times X_n &= \left[I_{m\times m}, T_{m\times(n-m)}\right] \times \begin{bmatrix} X_m \\ X_{n-m} \end{bmatrix} \\ &= \left[I_{m\times m} \times X_m\right] + \left[T_{m\times(n-m)} \times X_{n-m}\right] = X_m + Y_m^{'}. \end{aligned} \tag{9}$$

The final output consists of two components: the first part consists of the initial m bits from the input vector, while the second part is the result of multiplying the $m \times (n-m)$ Toeplitz matrix by the last $(n-m)$ bits of the input vector. We employ the algorithm outlined in Section 3.1.3 to compute $Y_m^{'}$. Additionally, we nearly halve the seed length by reducing it from $n+m-1$ to $n-1$ through a modified Toeplitz matrix. This adjustment enhances throughput and minimizes hardware overhead, as illustrated in Figure 5.

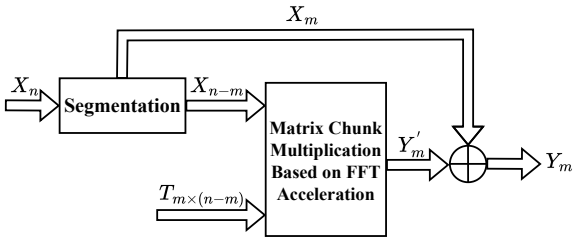

Figure 5: Post-processing using modified Toeplitz matrix.

3.2 The optimization of Trevisan extractor

The single-bit extractor proposed by this paper performs a two-stage randomness extraction process. The first stage uses a seed, denoted as s_1, and the second stage uses a seed denoted as s_2. The length of s_1 is $t_1 = \log_2(n)$. During the first stage, one random bit, referred to as x_{s_1}, is selected from the input sequence. The first stage then takes the l bits following x_{s_1} in the input sequence. These l bits are designated as $x_{sel} = \{x_{s_1}, x_{s_1+1}, \ldots, x_{s_1+l-1}\}$(if the index exceeds the sequence length, bits are taken from the beginning). The length of s_2 is $t_2 = l$. In the second stage, circular-shift inner products are performed using x_{sel} and s_2. We define $x_{sel(j)}$ as the result of the cyclically right-shifting x_{sel} by j bits, for $(1 \le j \le m)$. The output of the second extraction process is $\bigoplus_{i=1}^{l} x_{sel(1)}[i]s_2[i]$. Figure 6 illustrates the overall randomness extraction process of the single-bit extractor.

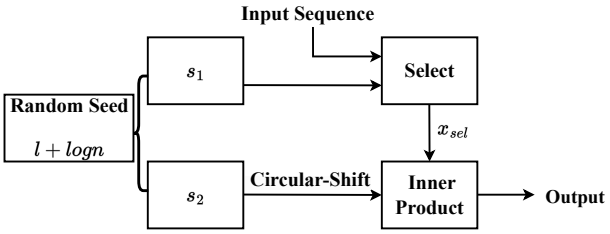

Figure 6: The computational process of the single-bit extractor.

The first process proposed in this work is a sampler and hardly changes the min-entropy density if we sample a random subset of bits from a weak random source [NZ96]. The seed length of this method is shorter than taking l bits from the input one by one, which needs a $l\log_2(n)$-length section of seed. Lu proposed another sampler by performing a random walk of l steps on a g-regular expander graph to reduce the seed length [Lu04] from $l\log_2(n)$ to $\log_2(n) + l\log_2(g)$. However, Lu's construction is not practical in real-world

applications, as the number of random walk steps increases considerably with decreasing input entropy density. Additionally, due to the complex structure of the expander graph, implementing this sampler requires substantial hardware resources, making it difficult to realize on FPGAs. An alternative approach to reduce the seed length is to use a linear corrector to compress the input length n to the sequence length l, which is the input length for the second extraction process. Unfortunately, the linear corrector can not handle inputs with both bias and correlation [ZB11] and it is challenging to compute the output entropy density when the input is biased and correlated. For the above reasons, we propose and utilize this sampling scheme that preserves the input entropy density while maintaining a short seed length.

The random extraction of our extractor is mainly realized by the second process, which is a strong blender with an output length of 1. This strong blender with inner product construction is shown as follows [DEOR04]:

$$\begin{aligned} BlE : \{0,1\}^n \times \{0,1\}^n &\to \{0,1\}^m \\ (x,y) &\mapsto ((A_1x)\cdot y, ..., (A_mx)\cdot y) \end{aligned} \tag{10}$$

where $\cdot$ is inner product mod 2 and A_ix is a matrix-vector multiplication over $GF(2)$. $A_1, ..., A_m$ are $n \times n$ matrices over $GF(2)$ so that for every nonempty subset $Q \subseteq [m]$, the rank of $A_Q \overset{def}{=} \sum_{i\in Q} A_i$ is at least $n - r$, for $0 \leq r \leq n$. The safety factor of this type of strong blender can be calculated as follows [DEOR04]:

$$\log_2\left(\frac{1}{\varepsilon}\right) = \frac{b_X + b_Y - (n + r + m)}{2} + 1. \tag{11}$$

b_X is the min-entropy of x and b_Y is the min-entropy of y. When A_i is a cyclic right-shift matrix and n is prime with 2 as a prime root, we have $r = 1$ [DEOR04]. In our extraction, we set $n = l, m = 1$ and $x = x_{sel}, y = s_2$. The second extraction performs a cyclic right-shift operation on the input vector x_{sel}, which is equivalent to multiplying the cyclic right-shift matrix. Thus, we have $\log_2\left(\frac{1}{\varepsilon}\right) = \frac{b_1+b_2-l}{2}$ to get the second seed length l.

b_1 is the min-entropy of x_{sel} and b_2 is the min-entropy of s_2. Since the random seed is uniform, we can get $b_2 = l$ and $b_1 = 2\log_2(1/\varepsilon)$. Therefore, this single-bit extractor is a $(2\log_2(1/\varepsilon),\varepsilon)$ strong extractor and a $(3\log_2(1/\varepsilon),3\sqrt{\varepsilon})$ quantum-proof strong extractor [MPS12]. As we have said, the input entropy density α is almost constant and we make an approximate estimate. Then we have $b_1 = \alpha l$ and $l = \frac{2\log_2(1/\varepsilon)}{\alpha}$. Let $\varepsilon' = 3\sqrt{\varepsilon}$, then this single-bit extractor is a $(6\log_2(3/\varepsilon'),\varepsilon')$ quantum-proof strong extractor. Therefore, the length of the random seed is shown by Equation (12).

$$t = l + \log_2(n) = \frac{4}{\alpha}\left(\log_2\left(\frac{3}{\varepsilon'}\right)\right) + \log_2(n). \tag{12}$$

3.3 Simulation of entropy sources

To verify the functionality of the post-processing module, we utilize the lightweight cryptographic algorithm PRESENT to simulate biased and correlated random bits as input. We then observe whether the output successfully eliminates these biases and correlations. The PRESENT algorithm enables the construction of a module that consistently generates uniform random numbers, serving as the foundation for generating imperfect raw random bits. It is important to note that the PRESENT algorithm is used here only for simulation purposes. In practice, many other cryptographic algorithms could achieve the same goal, with PRESENT being chosen as an example due to its lightweight implementation.

We can use look-up tables (LUT) to generate biased random bits. We generate a 2-bit biased random number based on the value of the fixed bit-width(e.g., 4-bit) ciphertext. Specifically, suppose that the value of the 4-bit ciphertext is in a specific interval (e.g.,

3-6). In that case, the output is noted as a corresponding value (e.g., 01), and the output is labeled with other values when the 4-bit ciphertext is in other intervals. Since the value of the 4-bit ciphertext is uniformly distributed from 0 to 15, we can freely set the output bias by adjusting the ciphertext comparison interval as shown in Figure 7. By definition, the min-entropy density of the input, in this case, is $H_\infty(X) = -\frac{\log_2(5/16)}{2} = 0.839$.

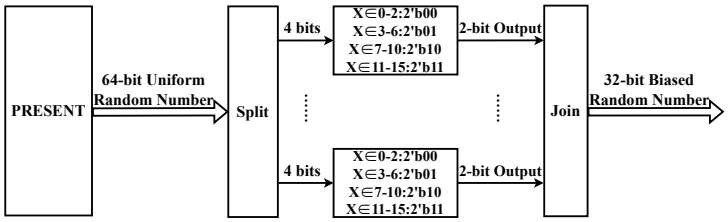

Figure 7: Generation process of the biased random numbers.

We use a Markov chain to generate correlated random bits by constructing all possible states, such as 00, 01, 10, and 11. These states are then transitioned according to a probability distribution defined by a fixed bit-width (e.g., 4-bit) ciphertext and the given probability transition matrix. The generating process is shown in Figure 8. Different probability transition matrices correspond to varying levels of serial correlation. To test our entropy extractor's ability to process both bias and correlation, we introduce bias into the correlated random numbers by adjusting the difference between the number of 0s and 1s in the probability transition matrix. Specifically, we set the maximum imbalance to 9 ones and 7 zeros (corresponding to a 4-bit state space). Using this matrix, we can calculate the conditional min-entropy as $H_\infty(X|Y) = -\log_2(9/16) = 0.83$, where X denotes the next state and Y denotes the current state.

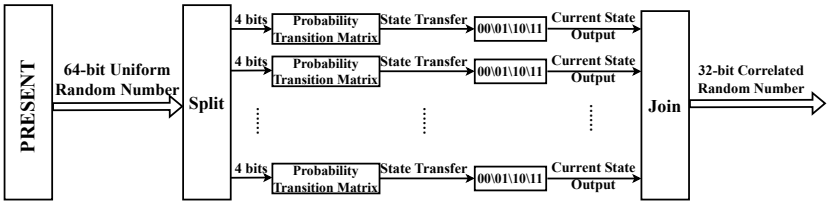

Figure 8: Generation process of the biased and correlated random numbers.

4 Hardware Implementation

To thoroughly evaluate the functionality of the two entropy extractors presented in this work, we implemented a ring oscillator-based elementary TRNG (ERO-TRNG) as the randomness source, generating non-perfect raw random numbers on the same FPGA.

The ERO-TRNG was proposed and modeled in [BLMT11], and it complies with AIS-31 standards. The generator utilizes two identical ring oscillators (RO1 and RO2) as sources of randomness. The output from one of the ring oscillators, a jittery clock signal, is sampled by a D flip-flop after an accumulation period. This period is determined by the second clock signal and is divided by a factor of K using a frequency divider [PMB+16]. The architecture of the ERO-TRNG core is illustrated in Figure 9.

As discussed in [PMB+16], the number of ring oscillator stages, N, is set to 5, and the value of K is adjusted from 20000 to 2000 to generate raw random numbers with insufficient entropy. The ERO-TRNG is implemented on the Xilinx Artix-7 FPGA, requiring only 19 LUTs and 26 FFs for its operation.

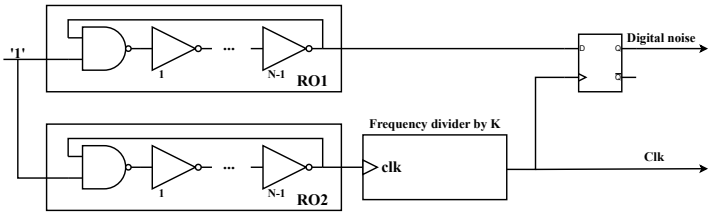

Figure 9: Architecture of the ERO-TRNG core.

4.1 Hardware implementation of the Toeplitz extractor

We design the simulation architecture of the Toeplitz extractor based on the algorithm described in Section 3.1, as shown in Figure 10. The raw random number simulator simulates the output of the entropy source. The n-bit raw random number is first fed into the FIFO. The FIFO takes out the first m bits and transmits them to the vector accumulation module. The remaining $n - m$ bits are output to the matrix multiplication module in chunks of k bits per cycle by the FIFO. The random seed simulator stores a binary sequence of length $n - 1$ from an entropy source and outputs a random seed with $m + k - 1$ bits to construct a $m \times k$ sub-matrix at each cycle. We must point out that the raw random number and random seed simulators are utilized only for simulation purposes. In a practical TRNG, the raw random numbers are directly derived from a physical entropy source, while the random seed is from another independent noise source with almost full entropy. The matrix multiplication module performs matrix-vector multiplication between a k-bit vector and a $m \times k$ sub-matrix, with FFT acceleration to enhance computation speed. Finally, the vector accumulation module sums all intermediate vectors and the first m bits of the raw random number after $(n - m)/k$ clock cycles to get the final result.

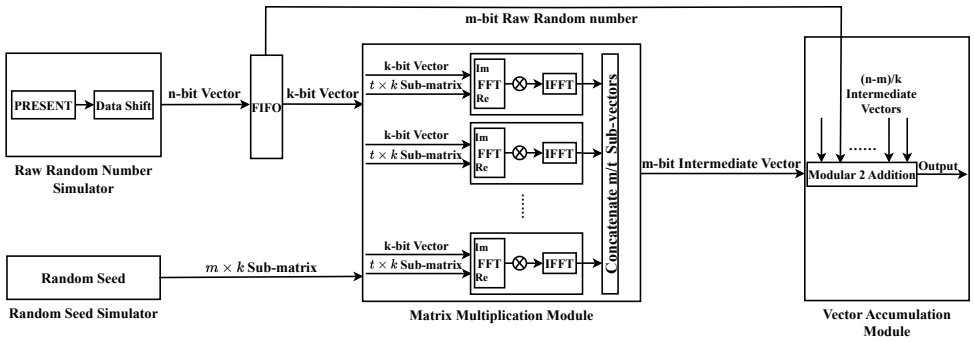

Figure 10: Simulation architecture of the Toeplitz extractor.

Considering the hardware overhead and the limitation of the FFT input length, we choose $n = 2304$, $m = 1728$, $k = t = 64$, and $\varepsilon = 2^{-80}$. According to Lemma 1, the algorithm can handle raw random numbers with entropy density larger than 0.82. Although the random seed of the Toeplitz extractor can be used repeatedly in theory, any entropy extractor that uses the same random seed multiple times makes the ε increase linearly in practice [FYEC25]. If we set the upper bound of $\varepsilon = 2^{-50}$ [LYL+21], we can perform $2^{-50}/2^{-80} \approx 10^9$ randomness extractions without changing the seed in our design.

By utilizing the modified Toeplitz matrix described in Section 3.1.4, only $(n-m)/k = 9$ cycles are required for each round of post-processing, as opposed to the $n/k = 36$ cycles needed without the modification. The matrix multiplication module also adopts a three-stage pipeline structure, as shown in Figure 11. The FFT module consists of 7 layers of butterfly arithmetic units, as the input length of FFT is $k + t = 128$. This configuration

requires 7 clock cycles to complete an FFT or IFFT operation. Additionally, the vector multiplication operation takes 1 clock cycle, and the entire matrix-vector multiplication process requires 15 clock cycles. After an initial 24 cycles, we obtain the first output. Due to the pipelined structure, an output of 1728 bits is generated every 9 cycles thereafter.

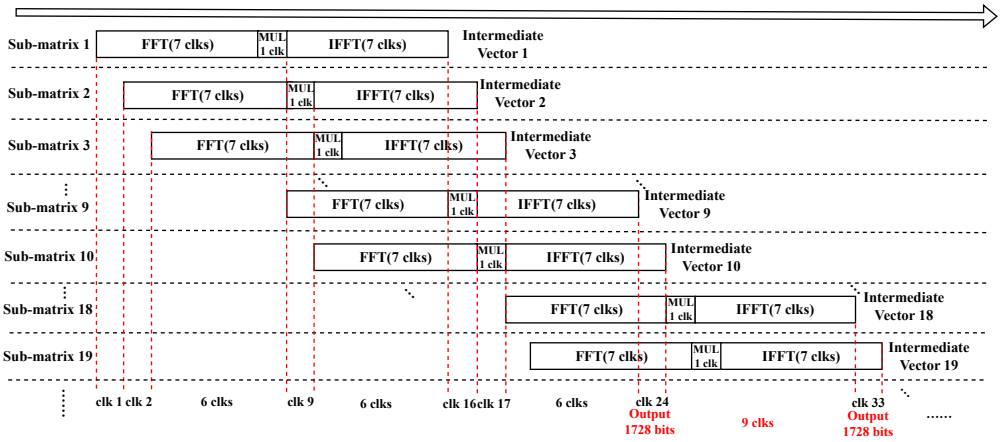

Figure 11: Pipeline for FFT-accelerated matrix multiplication.

The proposed Toeplitz extractor is implemented on the Xilinx Artix-7 FPGA. After timing optimization, the post-processing circuit operates at a clock frequency of 220 MHz, yielding a throughput of $1728 \times 220/9 \approx 42$Gbps. Performance comparisons of the Toeplitz extractor and other post-processings are summarized in Table 3. Due to the limited hardware resources available on the FPGA, the output length of our implementation is shorter than that of the GPU-based solution, but still longer than the outputs generated by the four other post-processing algorithms. Additionally, while our implementation consumes the most power due to the larger output length, this is not a major concern for high-performance applications, such as data servers, where power consumption is often a secondary consideration. The throughput achieved in this study significantly exceeds that of previous works, making it suitable for high-throughput true random number applications without excessive hardware resources. Our design delivers a six-fold performance improvement over prior Toeplitz extractor implementation [LLY+21] and an 84% increase in throughput-to-area ratio compared to most cryptographic post-processing solutions. Furthermore, our design exhibits the lowest compression rate, indicating minimal entropy loss. The Toeplitz extractor is capable of handling arbitrary correlations in raw random numbers, while most algorithmic post-processing methods can not. Eventually, our work offers unconditional security not based on any computational assumption or stochastic model, which is the core strength of our approach.

In addition to the FPGA implementation, we evaluate the efficacy of the Topelitz extractor through logic synthesis with a $45nm$ CMOS standard cell library. The performance metric for the Toeplitz extractor is shown in Table 4.

Table 3: Performance comparisons of the Toeplitz extractor.

Post-processing	This study Toeplitz+ERO	[LLY+21] Toeplitz	[ZNLZ16] Toeplitz	[YZY+22] SHA-2	[HJ12] AES	[RYDV15] XOR	[GV22] Linear code
Throughput	42 Gbps	6.8 Gbps	3.36 Gbps	0.681 Gbps	5.3 Gbps	0.0143 Gbps	0.0125 Gbps
Platform	Artix-7	GPU	Virtex-6	Virtex-7	Virtex-7	Spartan-6	Artix-7
Process Node	28 nm	16 nm	40 nm	28 nm	28 nm	45 nm	28 nm
Output Length	10^3	10^6	10^3	512	128	1	12
Slices	10527+11	—	—	413	2444	37	33
Block RAMs	0	—	—	9	0	0	0
kb/slice	3990	—	—	1649	2169	386	379
Power Consumption	1409 mW	—	—	125 mW	—	—	9.514 mW
Compression Rate	1.33	10	1.48	2	2	7	2
Output Entropy	0.999	0.999	0.999	—	—	0.999	0.999
Eliminating Correlation	✓	✓	✓	✓	✓	✗	✗
Unconditional Security	✓	✓	✓	✗	✗	✗	✗

Table 4: The performance metric for the Toeplitz extractor implemented on ASIC platforms.

Area	GE	Clock Frequency	Throughput
1034618.4 μm^2	1296514.3	1275 MHz	244.8 Gbps

4.2 Hardware implementation of the Trevisan extractor

In this section, we implement three different Trevisan extractors using the three single-bit extractors discussed in this paper and compare their performance. For this purpose, we utilize the unified weak design module. It is important to note that the general Trevisan extractor employs the Nisan-Wigderson pseudo-random generator as its weak design module. However, Raz et al. [RRV02] demonstrated that this is unnecessary. While the Nisan-Wigderson pseudo-random generator yields longer overlap lengths and reduced entropy loss, its complex polynomial calculations make it challenging to implement efficiently on FPGAs. As a result, existing Trevisan extractors that utilize this weak design are primarily implemented on PC or workstation platforms. In contrast, the weak design module proposed in this paper simplifies the process by splitting the seeds and overlapping the initial part of the latter section with the final part of the former section, as shown in Figure 12. This very simple method complies with the definition of weak design and requires minimal hardware resources. We label the ratio of the overlap length to the seed length of the single-bit extractor t as $\theta(\theta < 1)$. So we have $d = (1-\theta)mt + \theta t$ and $r = \frac{2^{\theta t}}{m}$ by definition.

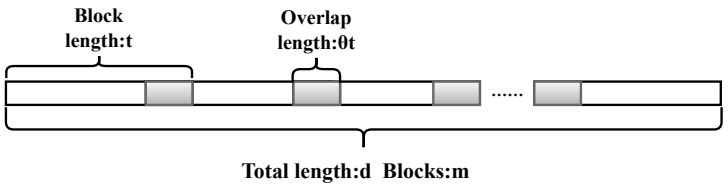

Figure 12: Schematic diagram of the weak design.

Figure 13 shows the simulation architecture of the Trevisan extractor. The raw random number and random seed simulators are similar to those implemented in the Toeplitz extractor, and both are employed for simulation only. The weak design module divides the random seed into segments and inputs each segment to a corresponding single-bit

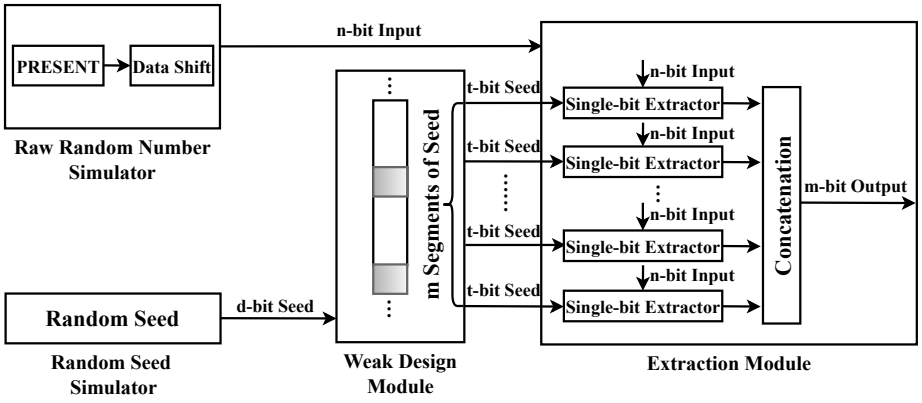

Figure 13: Simulation architecture of the Trevisan extractor.

extractor. The core of the system is the extraction module, which contains m single-bit extractors. Each single-bit extractor receives a seed of length t and an input of length n and outputs one random bit.

The parameters of the Trevisan extractor are calculated as outlined in Algorithm 1. Considering the introduction of parameters, we select an input length of $n = 512$ and an output length of $m = 8$. We assume that the input entropy density $\alpha = 0.82$, slightly lower than the entropy density of the raw random numbers we simulated to provide design margins. The number of times the seed is reused, $u = 2^{30}$, is set as an example, which matches the setting used in the Toeplitz extractor. The three quantum-proof strong single-bit extractors we introduced before also serve as inputs to the parameter calculation process. Beyond that, the output min-entropy density is required to be at least 0.98 according to the limitation of the 2024 version of AIS-31 [SP24].

The relation between output entropy density and ε is $H_\infty^{out} \geq -\log_2(2^{-m} + \varepsilon)$ where m is the output length. Taking into account the effect of reusing seeds, we can calculate that ε of the Trevisan extractor is $\varepsilon = (2^{-8\times 0.98} - 2^{-8})/2^{30} = 2^{-41.1}$ and we set ε to 2^{-42} for simplicity. Referring to Theorem 2, the safety factor of every single-bit extractor $\varepsilon' = 2^{-42}/m = 2^{-42}/8 = 2^{-45}$. We can then calculate the seed length t of the three single-bit extractors: 237 bits (for this study), 783 bits (for XOR-code), and 202 bits (for polynomial hashing). We also need to determine the overlap length of the weak design, which can be derived using Theorem 2 and Equation (2). Taking the single-bit extractor proposed in this work as an example to solve the equation in the 4th row of Algorithm 1, we traverse different input min-entropy densities, ranging from 0.7 to 0.95 [MPS12], and obtain corresponding relation curves. These curves are shown in Figure 14. Six specific points are labeled in Figure 14, with the overlap length ratios before each labeled point satisfying the required input min-entropy density. Based on our previous assumption of an input min-entropy density of 0.82, the maximum overlap length ratio θ that we can select is 0.0302. For design margins, we round this value to $\theta = 0.03$. Once θ is determined, we can proceed to compute the seed length d of the Trevisan extractor.

After determining all the parameters, we implemented the three relative Trevisan extractors on the Xilinx Artix-7 FPGA. The performance comparisons of three Trevisan extractors and the same four other post-processing methods are presented in Table 5. First, within the context of the three Trevisan extractors, the output rate of the polynomial hashing is the slowest. This is because the complexity of polynomial hashing and the limited hardware resources on the FPGA necessitate splitting the multiple rounds of finite-domain multiplication in a single extraction round, requiring 12 clock cycles for execution. The Trevisan extractor, utilizing the single-bit extractor proposed in this study,

Algorithm 1 Guideline for the calculation of the parameters of the Trevisan extractor.

Require: n, m α, number of times seed reused u, a (b, ε') quantum-proof strong single-bit extractor, and output entropy density $H_\infty = H_\infty^{out}/m = 0.98$ [SP24]

1: $\varepsilon = (2^{-mH_\infty} - 2^{-m})/u$
2: $\varepsilon' = \varepsilon/m$
3: Compute the seed length of single-bit extractor $t(\alpha, n, \varepsilon')$
4: Compute θ that satisfies the equation $\alpha = (b + 2^{\theta t(\alpha,n,\varepsilon')})/n$
5: d= $(1-\theta)mt + \theta t$.
6: **return** ε, θ, ε', t, d

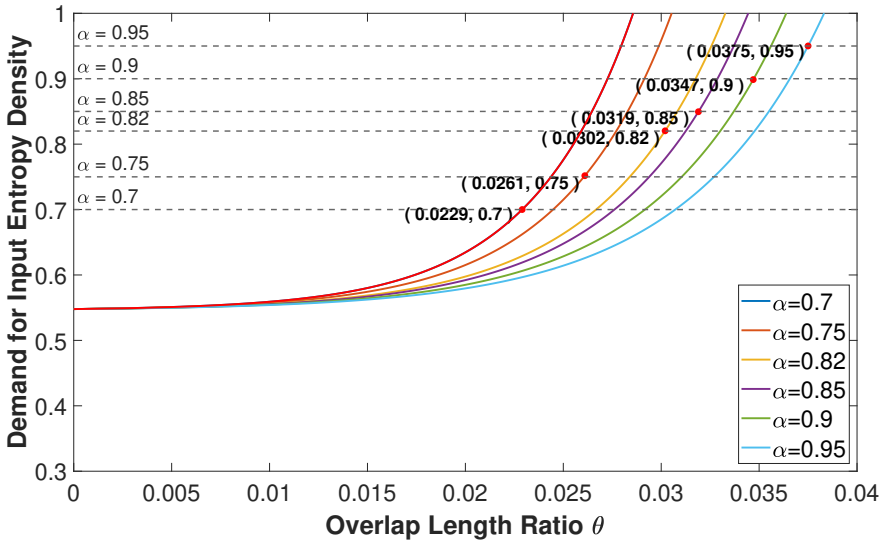

Figure 14: Effect of overlap length on demand for input min-entropy density.

demonstrates the lowest hardware overhead, the highest throughput-to-area ratio, and a shorter seed length among the three Trevisan extractors. Our design also achieves a 73% throughput-to-area ratio improvement compared to the XOR-code. Compared to most algorithmic post-processing techniques, the Trevisan extractor proposed in this paper still has significant advantages in both performance and applicability. While the overall performance of our work may not match that of cryptographic post-processing methods, the Trevisan extractor possesses the key advantage of unconditional security, similar to that of Toeplitz extractors. In contrast, cryptographic post-processing methods lack a formal proof of their entropy extraction capabilities.

Similarly, we conducted logic synthesis for the Trevisan extractor proposed in this paper with the same standard cell library. The performance metric is shown in Table 6.

It is important to note that, since the application scenarios discussed in this paper involve data centers or servers requiring high-speed random numbers, the two entropy extractors implemented exhibit higher power consumption, which is often a secondary consideration in these cases.

Table 5: Performance comparisons of Trevisan extractors.

	This study	XOR-code	Polynomial hashing	[YZY+22]	[HJ12]	[RYDV15]	[GV22]
Post-processing	Trevisan+ERO	Trevisan	Trevisan	SHA-2	AES	XOR	Linear code
Throughput	1.82 Gbps	2.35 Gbps	0.07 Gbps	0.681 Gbps	5.3 Gbps	0.0143 Gbps	0.0125 Gbps
Platform	Artix-7	Artix-7	Artix-7	Virtex-7	Virtex-7	Spartan-6	Artix-7
Process Node	28 nm	28 nm	28 nm	28 nm	28 nm	45 nm	28 nm
Slices	1656+11	3689	10185	413	2444	37	33
Block RAMs	0	0	0	9	0	0	0
Clock Frequency	227 MHz	294 MHz	105 MHz	278 MHz	456 MHz	100 MHz	—
kb/slice	1099	637	7	1649	2169	386	379
Power Consumption	542 mW	756 mW	2306 mW	125 mW	—	—	9.514 mW
Seed Length	237	783	202	—	128	—	—
Output Entropy	0.9891	0.9891	0.9891	—	—	0.999	0.999
Eliminating Correlation	✓	✓	✓	✓	✓	✗	✗
Unconditional Security	✓	✓	✓	✗	✗	✗	✗

Table 6: The performance metric for Trevisan extractor implemented on ASIC platforms.

Area	GE	Clock Frequency	Throughput
150362.1 μm^2	188423.7	825 MHz	6.6 Gbps

4.3 Guide to selecting appropriate post-processing methods

Building on the theoretical and implementation results previously obtained, this section provides a comprehensive overview of the key features of common post-processing methods, as illustrated in Figure 15[2]. It also offers general guidelines to assist TRNG designers in selecting the most suitable post-processing methods.

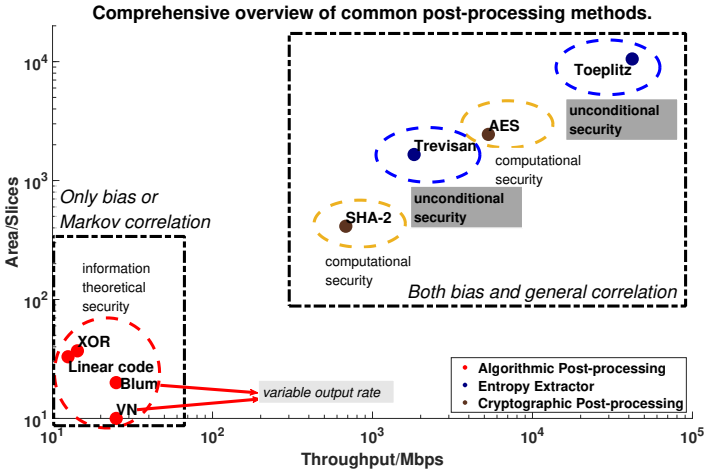

Figure 15: Comprehensive overview of common post-processing methods.

As shown in Figure 15, algorithmic post-processing is suitable for scenarios where raw random numbers exhibit minimal correlation and there is only one entropy source. In many applications that require a constant output rate, methods like the VN and Blum algorithms are not ideal due to their variable output rates. When the entropy source is of low quality, entropy loss becomes a critical issue, making proper linear code the optimal

[2]The throughput and area of the VN and Blum methods in the figure are approximate only.

algorithmic post-processing method. This is because alternative methods typically result in higher compression rates, which lead to lower output rates and greater entropy loss. On the other hand, when the entropy source is of high quality and hardware overhead is a primary concern, VN and XOR should be prioritized. These simple methods offer compact hardware implementations at the cost of reduced output rates.

In more cases where complex correlations are present in the raw random numbers, algorithmic post-processing methods are generally ineffective at eliminating these correlations. Therefore, cryptographic post-processing or entropy extractors are the viable alternatives. If cryptographic primitives are already integrated into the application system, lightweight cryptographic post-processing methods may be a suitable choice. However, the main drawback of these methods is their lack of formal information-theoretical proofs for entropy extraction, relying instead on the computational security of the cryptographic algorithms. For data-intensive applications, such as servers where speed is prioritized over area overhead and power consumption, or when raw random numbers originate from multiple correlated entropy sources, the entropy extractor implemented in this paper offers the optimal solution. Notably, both the Toeplitz and Trevisan extractors, whose security proofs are not based on any computational assumption or stochastic model, ensure unconditional security.

5 Post-processing test

We know that modern standards, such as AIS-31 [SP24], insist on formal analysis and a stochastic model for evaluating randomness sources. Specifically, AIS-31 mandates that designers submit a stochastic model of the physical entropy source for TRNG evaluation, ensuring that the proposed TRNGs are compliant with functionality classes PTG.2 and PTG.3 [SP24]. However, while statistical test results do not offer definitive proof, they can serve as a useful complement to assess the effectiveness of post-processing circuits.

We collect 1G bits of output data via UART. In practical applications, the output of the entropy extractor is the input of other modules, typically transferred through a high-speed interface. In this study, we just use UART for testing, with the post-processing module being suspended during data reception. In addition, we also collect the output data from an actual metastability-based entropy source with an entropy density of 0.725119 according to the NIST SP800-90B test. These data are then used as inputs to the two entropy extractors, and we evaluate the outputs.

We note that the use of NIST SP800-22 has been rejected for assessing cryptographic random number generators [oST22], but we just utilize these procedures to detect deviations of output sequences from uniformity [RSN+10]. Considering the potential obsolescence of the NIST SP800-22, we also use the NIST SP800-90B and AIS-31 test suites to further add credibility. We can find that the test results of the NIST SP800-22, NIST SP800-90B, and AIS-31 indicate the effectiveness of both post-processing circuits as shown in Table 7. In addition, we also utilize one publicly available file consisting of non-perfect random numbers from a real RO entropy source, which is *ringOsc-nist.bin* from the NIST SP800-90B. The test results once again validate the effectiveness.

Eventually, we collect and test the output data of the ERO-TRNG we implement. The test results of raw random numbers and processed random numbers are shown in Table 8. We can find that the raw random numbers from ERO-TRNG have fatal statistical defects, but the processed internal random numbers can pass all statistical tests. Although passing the black-box test is not conclusive evidence, it does support the validity of post-processing.

Table 7: NIST SP800-22, NIST SP800-90B and AIS-31 test on processing results.

	Topelitz/Trevisan	Topelitz/Trevisan	Topelitz/Trevisan
NIST SP800-90B	Biased Data	Biased and Correlated Data	Actual Entropy Source Data
Min-Entropy	0.99944/0.999584	0.999517/0.999441	0.998564/0.998628
Chi Square	Pass/Pass	Pass/Pass	Pass/Pass
Repeated Substring	Pass/Pass	Pass/Pass	Pass/Pass
IID Permutation	Pass/Pass	Pass/Pass	Pass/Pass
AIS-31	Biased Data	Biased and Correlated Data	Actual Entropy Source Data
P1:T0-T5	Pass/Pass	Pass/Pass	Pass/Pass
P2:T6a-T7b	Pass/Pass	Pass/Pass	Pass/Pass
P2:T8(Entropy≥7.976)	7.997462/8	7.996178/7.999759	8/7.999292
NIST SP800-22	Biased Data	Biased and Correlated Data	Actual Entropy Source Data
Frequency	0.868/100%/0.224/100%	0.74/98%/0.192/100%	0.401/98%/0.834/98%
BlockFrequency	0.35/98%/0.851/98%	0.658/98%/0.262/99%	0.475/100%/0.779/99%
CumulativeSums	0.911/99%/0.115/100%	0.225/98%/0.616/100%	0.699/99%/0.699/98%
Runs	0.35/99%/0.103/100%	0.575/98%/0.262/99%	0.213/100%/0.554/98%
LongestRun	0.335/100%/0.616/99%	0.868/100%/0.437/99%	0.596/98%/0.289/100%
Rank	0.384/99%/0.475/98%	0.202/99%/0.72/100%	0.145/100%/0.911/99%
FFT	0.172/98%/0.35/100%	0.846/99%/0.679/100%	0.289/100%/0.956/100%
NonOverlappingTemplate	0.699/99%/0.699/99%	0.834/100%/0.798/99%	0.249/98%/0.213/99%
OverlappingTemplate	0.437/98%/0.554/98%	0.35/98%/0.911/100%	0.898/99%/0.616/100%
Universal	0.779/99%/0.554/98%	0.192/99%/0.163/100%	0.816/100%/0.153/100%
ApproximateEntropy	0.319/100%/0.924/99%	0.637/98%/0.596/99%	0.367/98%/0.946/98%
RandomExcursions	0.671/99%/0.407/100%	0.401/100%/0.145/100%	0.350/100%/0.071/100%
RandomExcursionsVariant	0.383/100%/0.437/99%	0.72/100%/0.437/98%	0.74/100%/0.383/100%
Seria	0.494/100%/0.334/100%	0.437/99%/0.304/99%	0.163/100%/0.867/99%
LinearComplexity	0.851/99%/0.924/100%	0.72/99%/0.145/100%	0.383/98%/0.137/99%

Table 8: Test results of ERO-TRNG.

		Raw random numbers	Topelitz processed	Trevisan processed
SP800-90B	Min-Entropy	0.604165	0.998076	0.998322
AIS-31	T1 T2 T3 T6a	Fail	Pass	Pass
	T0 T4 T5 T6b T7a T7b	Pass	Pass	Pass
	T8(Entropy≥7.976)	0.30773	7.99946	7.99992
SP800-22	Frequency	0/0	0.956/100%	0.276/100%
	BlockFrequency	0/0	0.202/100%	0.304/100%
	CumulativeSums	0/0	0.249/100%	0.182/99%
	Runs	0/0	0.679/100%	0.555/100%
	LongestRun	0/0	0.699/98%	0.137/99%
	Rank	0.129/98%	0.350/99%	0.137/100%
	FFT	0/0	0.319/99%	0.679/100%
	NonOverlappingTemplate	0.554/97%	0.237/100%	0.335/100%
	OverlappingTemplate	0/0	0.172/100%	0.983/100%
	Universal	0/0	0.867/100%	0.834/100%
	ApproximateEntropy	0/0	0.319/98%	0.514/98%
	RandomExcursions	0/0	0.162/100%	0.213/100%
	RandomExcursionsVariant	0/0	0.401/100%	0.350/100%
	Seria	0/82%	0.172/100%	0.883/99%
	LinearComplexity	0.172/100%	0.817/98%	0.249/100%

6 Conclusion

This work implements two high-performance entropy extractors, the Toeplitz extractor and the Trevisan extractor, tailored for different high-speed application scenarios. We propose an architecture for the Toeplitz extractor that incorporates various optimization algorithms to improve the throughput. We also introduce a lightweight single-bit extractor to construct the Trevisan extractor. Both entropy extractors are implemented on the Artix-7 FPGA, achieving throughputs of 42 Gbps for one and 1.82 Gbps for the other.

Acknowledgments

The authors thank the editors and the anonymous reviewers of TCHES for their constructive comments. This work is supported in part by the National Key R&D Program of China (Grant No.2023YFB4403500, No. 2024YFB3108103), and in part by the National Natural Science Foundation of China(Grant No. 62274102). In addition, this work is partially supported by the European Commission through the Horizon 2020 research and innovation program (Belfort ERC Advanced Grant 101020005 695305), the DIGITAL-SIMPLE project Be-QCI (Contract 101091625), and the Horizon Europe Program (Grant 101114043 QSNP), as well as by the Flemish Government through the Cybersecurity Research Program (Grant VOEWICS02). We also thank Shuying Yin, who is a coworker at Tsinghua University, for synthesizing the circuits with the use of ASIC design tools.

References

[BLMT11] Mathieu Baudet, David Lubicz, Julien Micolod, and Andrié Tassiaux. On the Security of Oscillator-Based Random Number Generators. *Journal of Cryptology*, 24(2):398–425, April 2011.

[Blu86] Manuel Blum. Independent unbiased coin flips from a correlated biased source - A finite state markov chain. *Combinatorica*, 6(2):97–108, June 1986.

[CG85] Benny Chor and Oded Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 429–442, October 1985. ISSN: 0272-5428.

[CW79] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, April 1979.

[DEOR04] Yevgeniy Dodis, Ariel Elbaz, Roberto Oliveira, and Ran Raz. Improved Randomness Extraction from Two Independent Sources. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 3122, pages 334–344. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[DPVR12] 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, January 2012.

[DVK14] Siva Nishok Dhanuskodi, Arunkumar Vijayakumar, and Sandip Kundu. A Chaotic Ring oscillator based Random Number Generator. In *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 160–165, May 2014.

[FBDS23] Viktor Fischer, Florent Bernard, Quentin Dallison, and Maciej Skórski. Enhancing Quality and Security of the PLL-TRNG. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 211–237, August 2023.

[FYEC25] Cameron Foreman, Richie Yeung, Alec Edgington, and Florian J. Curchod. Cryptomite: A versatile and user-friendly library of randomness extractors. *Quantum*, 9:1584, 2025.

[GDS+20] Laurent Gantel, Alexandre Duc, Lucie Steiner, Fabien Vannel, Andres Upegui, and Florent Gluck. A FPGA-Based Post-Processing and Validation Platform

for Random Number Generators. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 123–126, May 2020.

[GGBB21] Vladimir Grozov, Alexei Guirik, Marina Budko, and Mikhail Budko. Construction of a Cryptographically Secure Pseudorandom Sequence Generator Based on the Blender Algorithm. In *2021 13th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 156–161, Brno, Czech Republic, October 2021. IEEE.

[GL89] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, STOC '89, pages 25–32, New York, NY, USA, February 1989. Association for Computing Machinery.

[GV22] Miloš Grujić and Ingrid Verbauwhede. TROT: A Three-Edge Ring Oscillator Based True Random Number Generator With Time-to-Digital Conversion. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 69(6):2435–2448, June 2022. Conference Name: IEEE Transactions on Circuits and Systems I: Regular Papers.

[Hay11] Masahito Hayashi. Exponential Decreasing Rate of Leaked Information in Universal Random Privacy Amplification. *IEEE Transactions on Information Theory*, 57(6):3989–4001, June 2011.

[HfWCl+12] Zhang Hong-fei, Jian Wang, Luo Chun-li, Lin Sheng-zhao, Zhou Lei, Hao Liang, Teng Chen, Kai Chen, and Pan Jian-wei. A real-time QKD system based on FPGA. *Journal of Lightwave Technology*, 30:3226–3234, October 2012.

[HILL99] 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, January 1999.

[HJ12] Ulfat Hussain and Habibullah Jamal. An Efficient High Throughput FPGA Implementation of AES for Multi-gigabit Protocols. In *2012 10th International Conference on Frontiers of Information Technology*, pages 215–218, December 2012.

[IJK09] Russell Impagliazzo, Ragesh Jaiswal, and Valentine Kabanets. Approximate list-decoding of direct product codes and uniform hardness amplification. *SIAM Journal on Computing*, 39(2):564–605, 2009.

[ILL89] R. Impagliazzo, L. A. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, STOC '89, pages 12–24, New York, NY, USA, February 1989. Association for Computing Machinery.

[Joh18] David Johnston. *Random Number Generators—Principles and Practices: A Guide for Engineers and Programmers*. De Gruyter, September 2018.

[KG04] Paul Kohlbrenner and Kris Gaj. An embedded true random number generator for fpgas. In *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays*, FPGA '04, pages 71–78, New York, NY, USA, 2004. Association for Computing Machinery.

[KT08] Robert T. Konig and Barbara M. Terhal. The bounded-storage model in the presence of a quantum adversary. *IEEE Transactions on Information Theory*, 54(2):749–762, February 2008.

[LLY+21] Yujie Luo, Yang Li, Jie Yang, Li Ma, Wei Huang, and Bingjie Xu. Optimization of the Randomness Extraction Based on Toeplitz Matrix for High-Speed QRNG Post-Processing on GPU. In *2021 13th International Conference on Communication Software and Networks (ICCSN)*, pages 261–264, June 2021. ISSN: 2472-8489.

[LSZZ23] Qian Li, Xiaoming Sun, Xingjian Zhang, and Hongyi Zhou. *Improved Real-time Post-Processing for quantum Random Number Generators*. ARxiv, January 2023.

[Lu04] Chi-Jen Lu. Encryption against Storage-Bounded Adversariesfrom On-Line Strong Extractors. *Journal of Cryptology*, 17(1):27–42, January 2004.

[LYL+21] Zhenguo Lu, Shenshen Yang, Jianqiang Liu, Xuyang Wang, and Yongmin Li. Efficient FPGA implementation of high-speed true random number generator. *Review of Scientific Instruments*, 92(2):024706, February 2021.

[MK23] Ahmed Maache and Abdesattar Kalache. Design and Implementation of a flexible Multi-purpose Cryptographic System on low cost FPGA. *International journal of electrical and computer engineering systems*, 14(1):45–58, January 2023.

[MPS12] Wolfgang Mauerer, Christopher Portmann, and Volkher B. Scholz. A modular framework for randomness extraction based on Trevisan's construction, December 2012. arXiv:1212.0520 [quant-ph].

[MXX+13] Xiongfeng Ma, Feihu Xu, He Xu, Xiaoqing Tan, Bing Qi, and Hoi-Kwong Lo. Postprocessing for quantum random-number generators: Entropy evaluation and randomness extraction. *Physical Review A*, 87(6):062327, June 2013.

[Neu51] Von Neumann. Various techniques used in connection with random digits. *National Bureau of Standards, Applied Math. Series*, 12:36–38, 1951.

[NHL+15] You-Qi Nie, Leilei Huang, Yang Liu, Frank Payne, Jun Zhang, and Jian-Wei Pan. The generation of 68 Gbps quantum random number by measuring laser phase fluctuations. *Review of Scientific Instruments*, 86(6):063105, June 2015.

[NZ96] Noam Nisan and David Zuckerman. Randomness is Linear in Space. *Journal of Computer and System Sciences*, 52(1):43–52, February 1996.

[oST22] National Institute of Standards and Technology. Decision to Revise NIST SP 800-22 Rev. 1a. https://csrc.nist.gov/news/2022/decision-to-revise-nist-sp-800-22-rev-1a, 2022. Accessed: 2025-04-15.

[PMB+16] Oto Petura, Ugo Mureddu, Nathalie Bochard, Viktor Fischer, and Lilian Bossuet. A survey of AIS-20/31 compliant TRNG cores suitable for FPGA devices. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–10, Lausanne, Switzerland, August 2016. IEEE.

[RRV02] Ran Raz, Omer Reingold, and Salil Vadhan. Extracting all the randomness and reducing the error in Trevisan's extractors. *Journal of Computer and System Sciences*, 65(1):97–128, August 2002.

[RSN+10] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, N. Heckert, James Dray, San Vo, and Lawrence Bassham. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. Technical Report NIST Special Publication (SP) 800-22 Rev. 1, National Institute of Standards and Technology, April 2010.

[RYDV15] Vladimir Rozic, Bohan Yang, Wim Dehaene, and Ingrid Verbauwhede. Highly efficient entropy extraction for true random number generators on FPGAs. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 1–6, San Francisco California, June 2015. ACM.

[RYDV16] Vladimir Rožić, Bohan Yang, Wim Dehaene, and Ingrid Verbauwhede. Iterating Von Neumann's post-processing under hardware constraints. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 37–42, McLean, VA, USA, May 2016. IEEE.

[SB15] Vikram B. Suresh and Wayne P. Burleson. Entropy and Energy Bounds for Metastability Based TRNG with Lightweight Post-Processing. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(7):1785–1793, July 2015.

[SJHB87] H. Sorensen, D. Jones, M. Heideman, and C. Burrus. Real-valued fast fourier transform algorithms. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(6):849–863, 1987.

[Sko16] Maciej Skorski. Evaluating Entropy Sources for True Random Number Generators by Collision Counting. In Lynn Batten and Gang Li, editors, *Applications and Techniques in Information Security*, Communications in Computer and Information Science, pages 69–80, Singapore, 2016. Springer.

[SMS07] Berk Sunar, William Martin, and Douglas Stinson. A Provably Secure True Random Number Generator with Built-In Tolerance to Active Attacks. *IEEE Transactions on Computers*, 56(1):109–119, January 2007.

[SP24] Werner Schindler and Matthias Peter. A Proposal for Functionality Classes for Random Number Generators. 2024.

[SV84] M. Santha and U. V. Vazirani. Generating Quasi-Random Sequences From Slightly-Random Sources. In *Proceedings of the 25th Annual Symposium onFoundations of Computer Science, 1984*, SFCS '84, pages 434–440, USA, October 1984. IEEE Computer Society.

[SV86] Miklos Santha and Umesh V. Vazirani. Generating quasi-random sequences from semi-random sources. *Journal of Computer and System Sciences*, 33(1):75–87, August 1986.

[SYR+23] Shuqin Su, Bohan Yang, Vladimir Rožić, Mingyuan Yang, Min Zhu, Shaojun Wei, and Leibo Liu. A Closer Look at the Chaotic Ring Oscillators based TRNG Design. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 381–417, March 2023.

[TBK+18] 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. Technical Report NIST SP 800-90b, National Institute of Standards and Technology, Gaithersburg, MD, January 2018.

[TDL+06] Hiroki Takesue, Eleni Diamanti, Carsten Langrock, M. M. Fejer, and Yoshihisa Yamamoto. 10-GHz clock differential phase shift quantum key distribution experiment. *Optics Express*, 14(20):9522–9530, October 2006. Publisher: Optica Publishing Group.

[TSSR11] Marco Tomamichel, Christian Schaffner, Adam Smith, and Renato Renner. Leftover hashing against quantum side information. *IEEE Transactions on Information Theory*, 57(8):5524–5535, August 2011.

[WZYG18] Xiangyu Wang, Yi-Chen Zhang, Song Yu, and Hong Guo. High-speed Implementation of Length-compatible Privacy Amplification in Continuous-variable Quantum Key Distribution. *IEEE Photonics Journal*, 10(3):1–9, June 2018. arXiv:1805.02372 [quant-ph].

[YRG+18] Bohan Yang, Vladimir Rožić, Miloš Grujić, Nele Mentens, and Ingrid Verbauwhede. ES-TRNG: A High-throughput, Low-area True Random Number Generator based on Edge Sampling. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 267–292, August 2018.

[YZY+22] Mingyuan Yang, Yemeng Zhang, Bohan Yang, Hanning Wang, Shouyi Yin, Shaojun Wei, and Leibo Liu. A sha-512 hardware implementation based on block ram storage structure. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 132–135, 2022.

[ZB11] Hongchao Zhou and Jehoshua Bruck. Linear extractors for extracting randomness from noisy sources. In *2011 IEEE International Symposium on Information Theory Proceedings*, pages 1738–1742, July 2011. ISSN: 2157-8117.

[ZBY+22] Yibo Zhu, Yiming Bian, Jie Yang, Yichen Zhang, and Song Yu. 21 Gbps Source-Independent Quantum Random Number Generator Based on Vacuum Fluctuations. In *2022 Asia Communications and Photonics Conference (ACP)*, pages 2140–2142, Shenzhen, China, November 2022. IEEE.

[ZNLZ16] Xiaoguang Zhang, You-Qi Nie, Hao Liang, and Jun Zhang. FPGA implementation of Toeplitz hashing extractor for real time post-processing of raw random numbers. In *2016 Ieee-Npss Real Time Conference (rt)*, pages 1–5, August 2016.