Extracting Randomness from Nucleotide Sequencers for use in a Decentralised Randomness Beacon

Darren Hurley-Smith
Royal Holloway, University of London
United Kingdom

Alastair Droop
University of York
United Kingdom

Remy Lyon
Veiovia Ltd.
United Kingdom

Roxana I. Teodor
Veiovia Ltd.
United Kingdom

ABSTRACT

This paper presents an investigation of nucleotide sequencing based random number generators, refutation of naive approaches to this problem, and a novel random number generator design based on the characteristics of nucleotide sequencers such as the Oxford Nanopore Technologies (ONT) MinION. Common issues include misunderstanding the statistical properties of nucleotide sequences and the provenance of entropy observed in post-processed sequences extracted from such data. We identify that the use of sequences, expressed as base-pair (ATCG) sequences, for random number generation is not possible. The process by which such sequences are observed and reported by scientific instrumentation, provide a means by which entropy associated with nucleotide sequences (or more correctly the act of observing and recording them) can be observed. We report a novel method of extracting entropy from the process of reading nucleotide sequences, as opposed to the nucleotide sequences themselves. We overcome the limitations and inherent bias of nucleotide sequences, to provide a source of randomness decoupled from biological data and records. A novel random number generator drawing on entropy extracted from nucleotide sequencing is presented with validation of its performance and characteristics.

ACM Reference Format:
Darren Hurley-Smith, Alastair Droop, Remy Lyon, and Roxana I. Teodor. 2024. Extracting Randomness from Nucleotide Sequencers for use in a Decentralised Randomness Beacon. In *The 19th International Conference on Availability, Reliability and Security (ARES 2024), July 30–August 02, 2024, Vienna, Austria.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3664476.3664480

ACM ISBN 979-8-4007-1718-5/24/07
https://doi.org/10.1145/3664476.3664480

1 INTRODUCTION

Random number generator design is a highly varied field, with many approaches, misconceptions, and specific requirements. Random number generators can be divided into two broad categories: Pseudo random number generators (PRNGs), and 'true' random number generators (TRNGs).

PRNGs rely on algorithms and system-supplied entropy to generate random numbers by an unpredictable process, though some researcher identify that seed material is therefore crucial to robustness [5, 9]. Random (and its non-blocking variant urandom), the Linux staple PRNG service, is a good example of this. It is a PRNG that uses a locally stored entropy pool to seed random number generation through a process of cryptographically secure hashing.

TRNGs exploit naturally unpredictable processes, which supplies entropy for use in either direct conversion to binary values or as raw material for a cryptographic or hash function.

ID Quantique has developed a range of NIST-certified security products incorporating optical quantum TRNGs [10, 20]. Nuclear decay is also a source of entropy: Americium and unstable Nickel isotopes are used in a variety of generators [17]. TRNGs always have some hardware element to them to harness and process the raw entropy, though this hardware may not be specifically designed for the purposes of random number generation.

We present an example of this latter class of TRNG. Using statistical tests of randomness, we have identified that nanopore-based DNA sequencing devices have unpredictable properties associated with the duration of signal events produced by ONT nanopore nucleotide sequencers. Further, we have developed a method for extracting entropy from log and data files generated by such devices, which may then be used to either seed a PRNG or as random output in its own right. Our long-term aim is to utilise high quality entropy from large numbers of ONT sequencers to provide viable inputs to a Decentralised Randomness Beacon (DRB).

1.1 Contributions

In this work, we contribute a thorough statistical analysis of nucleotide sequences, focusing on base-call data. We challenge previous claims regarding the randomness of encoded or hashed nucleotide sequences. We provide the first rigorous statistical analysis of nucleotide sequences and claims regarding their randomness. We aim to guide future researchers in avoiding unwarranted assumptions when using biological data as a source of randomness. Crucially, we propose a novel method for extracting randomness from the raw data output of nanopore-based sequencing technology, conducting a statistical analysis to confirm its suitability as a potential candidate for further development as a Random Number Generator (RNG). Our RNG design satisfies the following novel criteria:

- The proposed RNG uses real sequencing data, ensuring that no additional expensive and/or wasteful work is required to generate input sequences;
- It passes lightweight statistical tests of randomness, proving initial fitness for use and identifying required post-processing techniques to achieve randomness suitable for cryptographic applications;
- Demonstrates the viability of extracting noise from commercial sequencing technologies (ONT MinION, PromethION and GridION) as well as the general concept of nanopore-based sequencing technology, without reverse engineering or modification of hardware.

Our novel entropy extraction method, though tested with small output bitstreams and lightweight statistical tests, provides a valid proof of concept. It demonstrates that, despite flaws in many existing approaches, viable methods exist to extract useful noise from sequencing hardware. Additionally, our method breaks the link between extracted entropy and input data, preventing the derivation of sensitive genetic sequences from the output bitstream.

2 RANDOMNESS FROM NUCLEOTIDE DATA

Cryptographic applications demand high-quality randomness for unpredictable cryptographic key generation. One-time pads (OTPs) are particularly valuable due to their time-limited confidentiality features, enhancing confidentiality guarantees when keys match message lengths [15].

DNA and RNA have been explored as potential data sources for OTP cryptosystems [8]. Common encoding techniques convert base pair characters (A, T, G, and C) into 2-bit tuples. Li et al. [13] propose encoding each nucleic acid as a 2-bit tuple.

Researchers initially inject randomness by pseudorandomly selecting files and sequences for key mixing and expansion in DNA-based cryptosystems [7, 13]. Gearheart et al.[7] assume a uniformly distributed mix of free-floating nucleotides. Balanici et al.[2] enhance Li et al.'s proposal by mixing randomly selected nucleic-acids from many different files, further distancing output data from source material, particularly when sequences are from different species/genera.

A common misconception in the literature is that DNA and RNA are incompressible, likely stemming from misunderstandings of the incompressibility properties of proteins at the macro scale [16]. Many proposed cryptosystems leveraging nucleotide data assume incompressibility after shuffling nucleotide files, but we challenge these claims:

C1: DNA sequences are random and cannot be compressed.
C2: The large number of sequenced organisms ensures a great diversity of nucleotide sequences.
C3: By randomly selecting individual nucleic-acids at random indices within sequencing data, randomness of keys can be assured.

Researchers employing key mixing and matrix expansion on nucleotide sequences often observe results seemingly validating these assumptions. Li et al. [13] note that their outputs meet the Avalanche criterion, primarily due to post-processing nucleotide sequences. We posit that it is the mixing, shuffling, and hashing of sequences that provides the majority of observed randomness, and that nucleic acid sequences have little inherent randomness.

We challenge prevailing assumptions about the randomness of nucleotide sequences, identifying errors in prior works and emphasizing that observed randomness is a product of specific techniques. Nucleotide sequences are not inherently incompressible, and achieving uniformly distributed samples is challenging. Our study demonstrates that nucleotide sequences make a minimal entropy contribution to random number generation algorithms.

2.1 Statistical properties of nucleotide sequences

In this subsection, we explore the properties of 6 DNA-based randomness extractors. Each extractor produces an output sequence resulting from some combination of encoding, source-file mixing, and hashing. We do not perform matrix expansion or other methods of post-processing, to better identify how the assumptions stated in section 2 break down under scrutiny. We make the following assumptions: Nucleotide sequences are provided in the form of char representations of nucleic acids (ACGT); Random numbers generated must be primarily derived from these sequences - merging, shuffling, and hashing of data is allowed, but salting or other forms of entropy injection are out of scope; Researchers can only access the results of base-calling, no in-line process is considered at this stage.

Sample preparation for our DNA-based experiments is an involved process:

(1) Eukaryotic genome sequencing projects in the Short Read Archive (SRA) are identified;
(2) The first 1350 of 12,950 fast5 file IDs were extracted;
(3) Cutadapt v2.3 was used to trim adapter sequences to ensure only sample nucleotides of unpredictable length and composition are used in our extractor;
(4) Fast5 files were concatenated and split into 100 parts, then converted to text files containing $\approx$ 768 million char representations of nucleotide bases.

Prior to processing and testing nucleotide sequences, we first challenge the assumption that nucleotide sequences are random and incompressible. A simple statistical analysis of the appearance of specific nucleotides over our dataset reveals a substantial AT bias over CG. Specifically, the proportion of individual nucleic-acids over 10 unique sequences comprised of 1 million nucleic-acids each was: $A = 0.28822$, $T = 0.29120$, $C = 0.21033$, and $G = 0.21025$

This partially refutes the first claim (C1), that *nucleotide sequences are inherently random*. Unless biological samples are thoroughly screened and mixed for the specific purpose of providing uniformly distributed bases, adenine (A) and thymine (T) biases are expected. This is because in DNA, A always pairs with T and G always pairs with C. As a result, DNA-based sequences will express a bias with a correlated base. Nucleotide sequences, which include both DNA and RNA, express a variety of biases, with different species/genera expressing different ratios of A-T(U) and C-G [6]. The imbalance between A and T (above) is caused by random sampling when selecting indices from multiple base-call files in step 4 (above). This stage is identical to those observed in works by both Balanici et al. and Li et al. [2, 13]

To balance samples in such a way to offset the biases of individual sequences would require a sophisticated statistical analysis of the candidate dataset - a time-consuming and costly process when more effective sources of entropy exist [10]. Furthermore, the processes required to generate samples that meet the conditions required by Gearheart et al. [7] are prohibitively expensive, unless one uses only virtual DNA sequences. This is due to the highly variable bias expression of GC within samples [3].

To challenge claims C1, C2, and C3, we establish the following parameters by which a suitably random output derived from such data may be identified:

a) Outputs should not be predictable, even if the order in which the DNA sequences are used is known;
b) Produced numbers should reliably pass commonly accepted statistical tests of randomness in the published literature [4, 12, 19];

First, we generate a variety of RNG outputs based on 48 samples of Sheep DNA. Figure 1 demonstrates how techniques are progressively applied over samples, to better understand the points at which entropy is derived from, or injected into, nucleotide sequences.

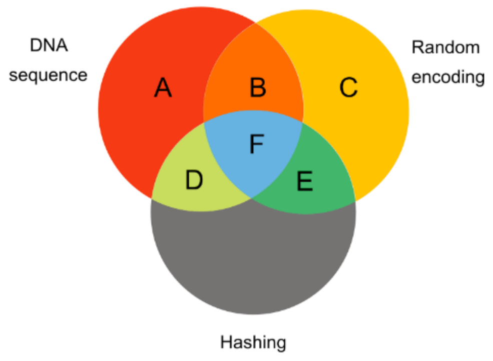

Figure 1: Encoding and hashing techniques applied over RNG permutations

Six RNG designs were trialled, aiming to generate output sequences that could pass statistical tests as rigorous as Dieharder, TestU01 Crush, and NIST SP800-22.

Figure 2 demonstrates the encoding and hashing operations performed over our dataset.

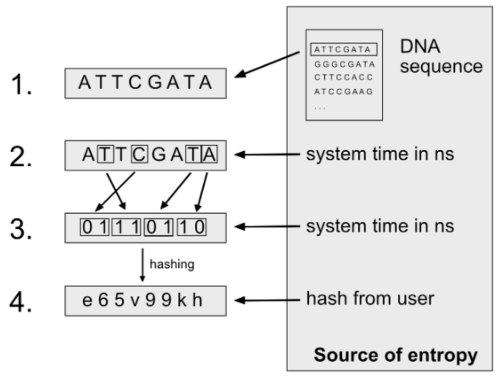

Figure 2: Stages of random number generation and sources of entropy

RNG-A utilizes DNA sequences without random nucleotide selection or further processing. RNG-B and F employ system

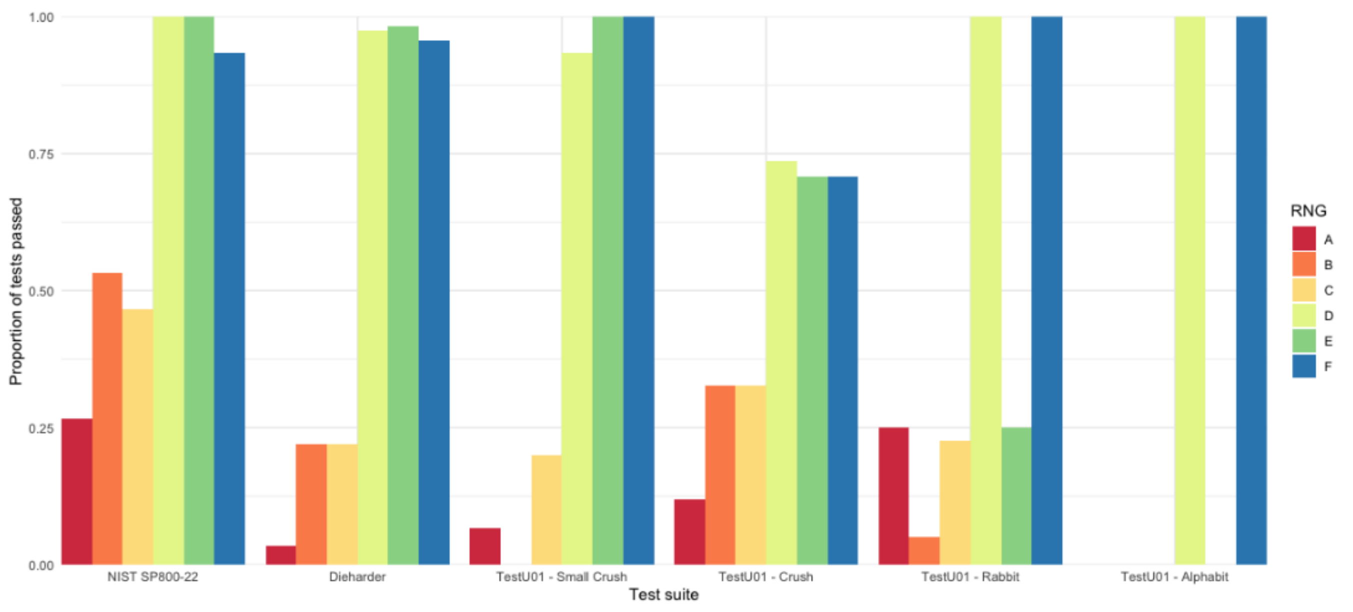

Figure 3: Pass rate of statistical tests over RNG outputs

time for random nucleotide selection and may randomize the encoding scheme per nucleotide using system time. RNG-D, E, and F apply hashing (SHA2-256) at any previous stage. RNG C and E uniquely use only randomly assigned 2-bit tuples from a pool, without employing DNA data. This establishes a set of control RNGs, enabling observations of the impact of encoding and hashing on both PRNG sequences and DNA. These RNGs generate 32-bit sequences after encoding but before hashing, represented as a *uint32* product of stages 1-3 (exit stage depending on RNG classification).

It is crucial to note that our prior criticisms of using DNA as a source of randomness are evident here. Attempts to extract randomness from DNA often result in its injection instead, even in contemporary works [18]. Shuffled outputs inherit entropy from the pseudorandom function used for selection, and hashing stretches this entropy by obfuscating the chaotic input data, assuming sufficiently long input strings to resist trivial Time Memory Trade-Off (TMTO) attacks. While DNA RNGs using hashing could be enhanced with pseudorandom salts, we do not explore this, as injecting more entropy does not contribute to the study of DNA's inherent properties as a source of randomness.

Figure 3 shows the results of 6 statistical test batteries performed over our 6 RNG designs.

2.1.1 NIST SP800-22 [19]. Each RNG underwent testing with a single 10Mb output for each of the 100 samples. RNG-D and E successfully passed all tests for every sample. RNG-F experienced notable failures in approximately 10% of the tested samples, primarily in the Monobits and Non-Overlapping Template Matching tests. These failures indicated a bias leading to correlations between 32-bit sequences in RNG-F outputs, possibly stemming from the acknowledged A-T bias in the DNA samples within our dataset. The intriguing observation is the failure of SHA2-256 to eliminate this bias for RNG-F, whereas RNG-D passes. This is due to biases in the 32-bit sequences used as input restricting the output domain of SHA2-256. In the more egregious cases, this leads to hash collisions, though the primary cause of failure in these tests was due to poor Hamming Distance test results, indicating repetitious structures within 32-bit sub-sequences.

The attributes of RNG-C that lead to failure are likely mitigated by hashing, elucidating why RNG-E performs exceptionally well despite utilizing the same randomized tuple selection method prior to hashing.

2.1.2 Dieharder [4]. All RNG designs exhibit failures, with RNG-D, E, and F displaying resilience despite an increased failure rate compared to NIST SP800-22 for D and E. RNG-F consistently fails the dab_bytedistribution and diehard birthdays tests, indicating an inherited bias and subsequent correlation from the source data.

Failures in RNG-D and E are caused by repetition of values (32-bit words) on an aperiodic, but structured, basis within sample files. While arbitrarily large sequences can be extracted using the proposed RNG designs, the limited pool of data available may result in biased or correlated sequences over repeated iterations of Dieharder.

For RNG-D, these failures arise from hash collisions when identical data is passed into the hashing algorithm, highlighting sensitivity to the complexity of DNA sequences. This aligns with our assertion that the structured nature of DNA necessitates careful shuffle and, ideally, source file mixing operations for suitability. DNA contributes minimal randomness, as illustrated by the consistent performance of RNG-A across all tests; it serves as a canvas for encoding and hashing operations to produce sequences inheriting entropy from those methods.

2.1.3 TestU01 [12]. TestU01 batteries of varied complexity assess our RNG designs, ordered by increasing rigour: Alphabits, Rabbit, Small Crush, and Crush. Due to the substantial data requirement of 84GB for Crush tests (and 0.8GB for Small Crush) additional samples were processed specifically for TestU01 Small Crush and Crush. The repetition of underlying DNA sequences, and bias inherited from the documented AT-pair bias, leads to repeated failures in all Crush test runs, with slight variations in the nature and proportion of failures. RNG E and F report a 0.74 pass rate, but fail tests of correlation and hamming distance. As RNG F is unsalted, hash collisions are observed in larger (Crush) test sequences.

A consistent failure in every run is observed in the LempelZiv test by RNG-A and D, indicating compressibility and challenging the notion of DNA's incompressibility and randomness presented in prior literature. RNG-D, characterized by hash collisions due to repeated identical inputs to SHA2-256, also fails this test.

Small Crush, a less demanding battery requiring only 0.84GB of data, sees RNG-E and F passing, highlighting the positive impact of hashing on the outputs of both DNA and PRNG-selected 2-bit tuples.

RNG-A performs as expected, displaying poor performance due to the compressibility of test sequences and a high degree of correlation between 2-bit tuples due to A-T and G-C pairwise biases. Surprisingly, RNG-B exhibits exceptionally poor performance due to biases amplified by the bias-agnostic encoding scheme used. RNG-C shares this issue to a lesser extent, experiencing periodic runs of 2-bit tuples caused by the PRNG itself.

Alphabits and Rabbit, lightweight tests suitable for hardware implementation, demonstrate RNG-D and F passing every test for all sequences, underscoring the effectiveness of hashing in stretching entropy given sufficient diversity between provided subsequences.

RNG-A, B, and D consistently fail every iteration of the randomwalk1 test, indicating a bit-level bias, specifically a trend towards 0 or 1, as observed in both Rabbit and Alphabits.

2.2 Refutation of prior claims.

We can refute the previous claims regarding the suitability of data representing sequences of nucleic-acids for random number generation as follows:

C1: **DNA sequences are random and cannot be compressed.** RNG-A consistently highlights the high compressibility of raw DNA sequences, which can be attributed to the inherent biases of DNA biochemistry. RNG-D, demonstrates that hashing improves the randomness of nucleic-acid sequences, but rigorous tests (Dieharder, NIST-SP800-22, and Crush) identify repetitions in the 32-bit sequences used as input to SHA2-256. RNG-F exhibits identical issues, demonstrating that the output of such methods is unsuitable for use as a seed to a PRNG or input to a hash function fulfilling a similar role.

C2: **The large number of sequenced organisms ensures a great diversity of nucleotide sequences.** Despite generating 100 outputs from 1350 randomly selected fast5 files, DNA-based RNGs (A, B, D, and F) exhibit suboptimal performance in rigorous test batteries (Crush, NIST, Dieharder). Consequently, authentic DNA and RNA cannot substantiate this claim. The inherent bias of nucleotides is expressed among sufficient genii that it is impractical to expect that it will not influence the randomness of even shuffled sequences.

C3: **By randomly selecting individual nucleic-acids at random indices within sequencing data, randomness of keys can be assured.** This claim is refuted by the failure of RNG-B, D, and F in stringent test batteries. The application of encoding and hashing techniques fails to guarantee the elimination of inherent biases in tested sequences. We find that nucleotide data exhibits excessive correlation and bias, rendering it unsuitable for cryptographic key material without the infusion of substantial entropy through supplementary processes.

We demonstrate that use of DNA is deleterious to the subsequent use of hashing to produce apparently random numbers. DNA is wholly unsuitable as a source of randomness, even when shuffled using a PRNG. However, the process of sequencing DNA using nanopore sequencers does produce usable entropy in the form of pico-amp signal noise, and temporal characteristics of nanopore signal outputs.

Privacy is crucial when handling sensitive genetic sequences, which may contain intellectual property or personal data. Even mixing base-called data and hashing output binary sequences may not prevent attacks, such as using Rainbow Tables to reverse less sophisticated hashing algorithms like MD5. Inversion of mixing functions would then allow the derivation of binary-encoded bases, potentially revealing input genetic data. This could lead to the recovery of the original genetic sequences. While cryptographic methods exist to prevent this, it is safer to decouple entropy extraction from meaningful data entirely.

The inherent bias and privacy issues surrounding the use of DNA itself as a source of randomness undermine any perceived utility in currently proposed methods of extracting entropy from base-called sequences. We propose a method that extracts noise inherent in the sequencing process, which is separate and uncorrelated with base-called sequences. This ensures that genetic data cannot be recovered even if the entropy extraction function is inverted.

3 NANOPORE-SEQUENCING HARDWARE BASED RNG

Based on our refutation of claims that one can use nucleotide sequences as a direct source of randomness, we have developed a novel technique to extract randomness from useful sequencing activities. Instead of proposing that sequencing is performed specifically for the purposes of generating banks of genetic data suitable for use as a source of randomness, we proposed a design that:

- Draws on underlying electro-chemical, optical, or electrical properties of nucleotide sequencing hardware;
- Requires no reverse engineering of software or hardware, utilizing diagnostic or pre-processed data to identify sources of entropy;
- Discards biological data (which is too highly structured to be of use) and instead focuses on the use of jitter, noise, and other unpredictable phenomena that manifest in raw signal outputs from sequencing technologies.

We focus on leveraging the electrical signal (charge expressed as pA) of nanopore-based sequencing hardware, as a source of randomness. By focusing on extracting entropy from useful sequencing tasks, our generator design can be used as an in-line process, allowing for generation of random numbers alongside usable biological data, while severing all informational links between the two outputs if required.

3.1 Data collection

We use fast5 files derived from ONT nanopore-based sequencers in this work. Nanopore-based nucleotide sequencers work by reading a change in ionic flux across a membrane as a nucleic acid is passed through a protein channel in the membrane [14]. This methodology directly "reads" the molecule rather than relying on observation of strand-complimentary synthesis of strands. This difference provides a great advantage in terms of speed and possible sequence length, although the resulting signal is more complex to interpret than traditional sequencing methodologies. Most such devices are comprised of an array of nanopores, each under potentially different thermal, electro-osmotic pressure, or electronic resistance conditions. Furthermore, the duration for which a nanopore remains functional in any device is variable. These are all factors that may contribute hypothetically random elements to the raw signal output produced by such devices.

The raw data produced by such devices consists of current (pA) measurements across the membrane as the polymer of interest moves through the pore. Analysis of these raw signal values provides two possible sources of randomness: the time taken for a single polymer (a single read) to pass through the channel, and the idle duration between two successive reads passing through the same channel. We refer to these two times as the *read time* and the *gap time* respectively. *Gap time* is a function of both the underlying (real) gap observed by the nanopore-based sequencing hardware, and combined hardware/software attempts to identify the presence of a read. This may introduce error, possibly manifesting as a fuzzy bound, to the *gap time* calculation. Other sources of randomness, such as jitter in the current-read, or the dwell-time of specific states in the signal output, are potential sources of entropy. However, we defer such investigations to a subsequent body of work.

Raw signal data is collected on the device during a sequencing run, and is subjected to initial preprocessing. This identifies the start and end of each read, allowing the device to output time-stamped raw data traces for each individual read. Reads are tagged with the channel through which they travelled, thus allowing extraction of both the read and gap times (expressed in milliseconds with a resolution of 0.2 ms) for the device channels.

We take the fast5 output of ONT devices, and reformat the output to produce a space-delimited list of integers expressing intervals differentiated as gaps or reads to simplify processing data during the entropy extraction process. Our post-processed files begin with a gap at index 0 followed by a read at index 1.

3.2 Randomness extraction from nucleotide sequencing timing data

The data produced by the method detailed in sub-section 3.1 requires further processing to produce a hypothetically random bitstream. We separate gaps from reads to create two sub-files for each sample, as reads trend towards consistent duration (ms) throughout the majority of a sample. This leads to highly correlated output, while gap-only sub-files demonstrated significantly higher variance. We do not explore the characteristics of read data in this work, due to its poor performance in basic tests of randomness (FIPS 140-2) regardless of extraction method applied. These characteristics are discussed further in Section 4.

Table 1: ent test results for lag extraction.

File	Bytes	Entropy	χ^2	p-value	Arith. Mean	MC π	Serial Corr.
S1	218435	7.999146	259.00	0.4185	127.0573	3.154951243	-0.002844
S2	242148	7.998887	374.37	< 0.01	126.4569	3.162594777	-0.000639
S3	259698	7.998931	384.18	< 0.01	126.3946	3.171314373	0.002524
S4	427693	7.999054	560.88	< 0.01	126.3828	3.161920260	-0.001976
S5	426143	7.999090	537.41	< 0.01	126.3069	3.175478366	0.000073

Two methods of extracting randomness were tested:

- Compare lags between events in the sample data to produce an output bitstream.
- Compute modulo 2 over each scalar value associated with an event, and uses the output integer as the basis for bitstream generation.

Extracting lags was the original prototype approach, using timing data from nanopore-based sequencing as an array, over which a sliding window of gap length comparisons was performed. However, this method was found to preserve underlying biases. Specifically, the increasing size of gaps towards the end of a sample file (due to depletion of nucleotide materials in the fluid suspension) led to runs of 1 in excess of tolerable limits. This is amply demonstrated by the *ent* test suite, as shown in Table 1. χ^2 scores in excess of 310, high error in montecarlo π results, and serial correlation in the order of between 10^2 and 10^{-2} indicate underlying structure that precludes this extractors output being considered random. As a result, the lag-extractor was rejected in the early experimental stage.

By simply computing modulo 2 over an array of integers representing event durations, a marked improvement in efficiency and output quality can be observed. It avoids the correlation issues between reads and gaps by instead checking whether event durations are odd or even (l modulo 2), as shown in Algorithm 1:

Algorithm 1 Nucleotide sequence Modulo Extraction Algorithm

```
for $i \in f$ do
    if $(x[i] \mod 2) == 0$ then
        $b \leftarrow 0$
    else if $(x[i] \mod 2) \neq 0$ then
        $b \leftarrow 1$
    end if
    $v.append(b)$
end for
```

This method discards distinctions between discrete event definitions, as discrete events (x) are not compared based on magnitude, but by the output of $d \mod 2$. Even timings result in a 0 and odd results in a 1. This binary bit is then appended to an output vector (v) which is written to a binary file once all lengths have been processed. Output data must be trimmed to the nearest whole byte if it doesn't conform to a file where $(f \mod 8) == 0$. This culling of trailing bits prevents the introduction of bias through padding of the output data with arbitrary bits.

Algorithm 1 yields 418KB in output bits from a 9.4MB source file (itself derived from fast5 files that may be several GB in size). This represents an approximate compression rate of 20:1.

To verify the suitability of modulo extraction over gaps as a basis for use in a random number generator drawing on nucleotide sequencing methods as an entropy source, statistical tests of randomness were used as an initial check of output quality. We test 24 binary files, generated using the modulo extractor method, and a further file, which is the concatenation of the previous sample files. The concatenation of all outputs is intended to help determine whether individual samples, with variable lengths, can be combined to produce larger output sequences without loss of entropy. Section 4 details the statistical test results for a selection of gap files.

4 ANALYSIS AND DISCUSSION

4.1 Statistical tests of randomness

To identify whether the methods proposed in 3 are sufficiently random, the following statistical tests were applied:

- ent. [1]
- FIPS 140-2.
- TestU01: Alphabits and Rabbit batteries. [12]

More sophisticated tests of randomness, such as Dieharder [4] and NIST SP800-22 tests [19] require more data than individual runs (fast5 files) can provide. As this work represents an initial exploration of the properties of single runs of nanopore-based sequencers as a potential source of randomness, such tests are deemed out of scope for this paper.

4.2 FIPS 140-2

Table 2 shows the failed tests for FIPS140-2. We test against a confidence interval of $a \geq 0.99$, considering any pass rate below this threshold a failure. Modulo extraction over the

Table 2: FIPS140-2 Number of failed tests per Modulo Extracted Sample.

File	Pass	Mono.	Poker	Runs	L. Run	C. Run
S7	0.99	0	0	1	0	0
S10	0.99	0	0	0	1	0
S14	0.99	0	0	1	0	0
S15	0.99	0	0	0	1	0
S17	0.99	0	0	1	0	0
S19	0.99	1	0	0	0	0
S21	0.99	0	0	0	1	0
S22	0.99	10	2	1	0	0
S23	0.90	1	0	0	0	0
S_concat	0.99	15	2	0	3	0

listed samples passed FIPS 140-2 with one exception: S23. Out of 25 samples, 10 fail at least 1 test, though for 8 of these (all except S23 and S_concat) only one 20,000-bit subsequence fails in each case. S23 has an excess of 1's relative to 0's in tested bitstreams. As a result, it fails the monobits test repeatedly, with 1 overlong run of 1's. It also fails the Poker test twice, indicating a weighted distribution of 4-bit tuples in 2 of the tested 20,000-bit sequences for this file. S_concat, being a concatenation of all sequences, inherits these failing sequences and accumulates further borderline failures from the dataset, but does not fail the proportional test.

As previously noted, these are very simplistic tests, and are only recommended as a test for total failure. We use ent and TestU01 to further explore the characteristics of these sequences.

4.3 ent

Ent allows us to observe specific characteristics of each sample. Shannon entropy, a simple measure of entropy based on the expression of symbols within a set, is > 7.99 for all samples. However, a counter increasing monotonically to its maximum value, and looping in excess of $l = c^c$ times will report perfect Shannon entropy, so it is not a reliable means of determining randomness [11]. Compressibility, the degree to which a file's size can be reduced using common compression algorithms, is 0 for all files. This is a vast improvement over highly compressible nucleotide sequence encoding.

Arithmetic mean is a statistic that should represent the central tendencies of a sample. However, it is highly sensitive to outliers, which makes it an ideal test to identify whether values in a set are *identically distributed*. The degree of error from the expected mean (127.5 for 256 symbols - which represents single-byte encoding) indicates the degree of deviation from an *identical distribution*. A limitation of this test is that it cannot be used to observe whether values are *independently distributed* as it merely observes the mean value of a set, not its composition. The maximum error among our samples was 1.01% for S23. This sample was notable for it's failure of the FIPS 140-2 battery. All other samples pass the Arithmetic mean test.

Table 3: χ^2 and Serial Correlation results for modulo extraction.

File	Bytes	χ^2	p-value	Serial Corr.
S1	218869	237.52	0.7772	-0.003336
S2	242162	284.71	0.09	0.001745
S3	259711	257.18	0.45	-0.000688
S4	427740	292.22	0.05	0.000127
S5	426186	286.87	0.08	-0.001448
S6	983538	320.51	< 0.01	-0.001830
S7	1055980	281.69	0.12	-0.000614
S8	1127814	239.80	0.74	-0.001270
S9	438116	298.69	0.03	-0.000912
S10	1095264	225.64	0.91	0.000395
S11	202285	290.52	0.06	-0.001985
S12	573411	271.45	0.23	-0.001473
S13	613767	209.17	0.98	0.000952
S14	1758005	247.43	0.62	0.000357
S15	1525114	258.87	0.42	0.001267
S16	196847	249.02	0.59	0.001099
S17	509225	288.71	0.07	-0.003296
S18	134953	229.17	0.88	0.000348
S19	221714	309.56	0.01	0.000040
S20	267490	392.50	< 0.01	-0.001308
S21	491693	246.58	0.64	0.001561
S22	204288	321.10	< 0.01	-0.000517
S23	261631	557.864164	< 0.01	0.001425
S24	264387	338.626479	< 0.01	-0.000795
S_concat	11925522	230.58	0.86	-0.000213

Table 3 shows the χ^2 and serial correlation results for our samples. The majority of files (75%) pass ent, but 4 of those files which fail, do so due to the χ^2 test. All 5 failing files do so due to an over-representation of specific bytes. There is no discernible bias shared between the 5 files, however all failing files exhibit a bias towards 1's (0.506 of bits in each file) during bit-level χ^2. This indicates that some samples will inherit a bias from the signal-to-timings stage of processing. Further analysis of the software used to produce these timings will be required to establish a specific cause. In this case, gaps tend towards odd rather than even values. A second notable observation is that serial correlation appears completely decoupled from χ^2 and other statistics produced by ent. All serial correlation statistics are within tolerance (+/-0.5 would indicate non-randomness analogous to prose or C code).

Monte Carlo π plots successive 24-bit values as X and Y coordinates on a square grid. Every coordinate 'hit' by a value, within a circle defined within the grid, increments a counter used to count the proportion of hits. In a randomly distributed sample, the percentage of hits can be used to calculate π. The degree of error indicates the deviation from *identical distribution.* The largest error in our sample set is 0.6%, well within tolerance. The mean error is 0.2% and the media in 0.25%.

These results demonstrate the modulo extractor is a substantial improvement over both the methods explored in Section 2, and the lag extractor rejected in the early experimental stage. Having completed testing using basic tests for total failure, and having analysed basic sequence properties through ent, TestU01 was used to test the outputs with a more robust set of statistical tests.

4.4 TestU01

Table 4 shows the failed tests for the alphabits battery of TestU01. This lightweight test battery includes 9 tests over the first 1,000,000 bits of each file. As this would result in the concatenation of all files reporting the same results as S6, we omit the concatenated file from these results. The same principles apply to our Rabbit test results.

Table 4: Failed Alphabits Tests

File	Num Fails	Failed Test IDs
S6	1	6
S19	9	1, 2, 3, 5, 8, 9
S20	11	1, 2, 3, 5, 6, 8, 9
S21	8	1, 2, 5, 8, 9
S22	11	1, 2, 3, 5, 6, 8, 9
S23	11	1, 2, 3, 5, 6, 8, 9

Of the 25 tested files, 4 files identified as weak by *ent* continue fail multiple tests. One borderline file, S19, also fails by a significant margin (a total of 9 test failures over 6 of the 9 test categories). In total, 6 out of 25 files fail the Alphabits battery on at least 1 test. The most commonly failed tests are Multinomial BitsOver, Hamming Independence, and RandomWalk tests. The latter of these is an intuitive failure. Failure of the χ^2 test indicates that a bit-level bias is present, which these results confirm. Furthermore, Hamming Independence failures indicate that both 16 and 32-bit sequences have dependencies that indicate underlying structure, reinforcing both the ent and RandomWalk observations. The failure of specific Multinomial tests (1) by all failing samples further indicates an underlying bias in the distribution of 2-bit tuples.

Table 5 shows the failed tests identified for Rabbit. This battery is more intensive than Alphabits, applying 26 categories of test with multiple parameters per test. All samples which fail Alphabits appear in these results, but are joined by a further 6 samples (a total of 12 failing samples).

Table 5: Failed Rabbit Tests

File	Num Fails	Failed Test IDs
S1	1	24
S2	1	24
S2	1	20
S6	1	16
S11	1	1
S15	1	1
S19	12	11, 15, 20, 24, 25, 26
S20	16	11, 15, 16, 17, 20, 24, 25, 26
S21	1	1
S22	10	1, 11, 15, 20, 24, 25, 26
S23	17	11, 14, 15, 16, 17, 20, 24, 25, 26
S24	14	11, 14, 15, 16, 20, 24, 25, 26

Though the specific test failures are different, the underlying cause remains the same as that observed when evaluating Alphabits results. Failing samples tend to fail RandomWalk, Run, Multinomial, and Hamming Independence tests.

These failures indicate that further development of the post-processing method applied over device outputs is required. As the modulo extractor does not enforce Independent and Identical Distribution (IID), but instead observes it as an emergent property of the randomness of $l \mod 2$ within the timings of gaps and reads in DNA reader outputs, it is likely that it is sensitive to default outputs or steady states in the nucleotide sequencer's output (as expressed in fast5 files).

Table 6 shows the proportion of test battery passes (all tests passed) for Alphabits and Rabbit. RNGs A, B, D, and F are compared with raw, hashed, and salted & hashed output from the modulo extractor proposed in this work.

Raw modulo extractor output outperforms both encoded (A) and randomly encoded (B) nucleotide data by a substantial margin, demonstrating the higher inherent entropy of nanopore-based sequencing timing data. RNGs D and F, as well as hashed (unsalted and salted) modulo extractor outputs, report perfect pass rates for Alphabits. The modulo extractor outputs infrequently fail the Multinomial BitsOver test (Rabbit).

We identify that a small subset of sample files generated using the modulo extractor retain highly predictable timings, likely from device initialisation and residual adapter sequences. Removing the first 4,092 bytes from hashed modulo

Table 6: Comparison of TestU01 Alphabits and Rabbit results for selected RNGs

Generator	Type	Alpha	Rabbit
RNG-A	nucleotide data	0	0.25
Mod_raw	raw mod data	0.75	0.5
RNG-B	random nucleotides	0	0.05
RNG-D	hash RNG-A	1	1
RNG-F	hash RNG-B	1	1
Mod_hash	hash mod data	1	0.88
	4KB pruned data	1	1
Mod_salt_hash	hash + salt mod data	1	0.92
	4KB pruned data	1	1

outputs results in a 100% pass rate for hashed modulo outputs. This informs us that in the case of real-time generation of random numbers from nanopore-based sequencing timings, initialisation data must be discarded, even if it appears random in over 80% of cases. This prevents the introduction of biases caused by device and/or sample state. This pruning process does not benefit raw modulo outputs, though failure rates for samples reports in Tables 4 and 5 are reduced.

4.5 Discussion

It must be noted that the samples presented in this work are the unfiltered output of ONT nanopore-based sequencing devices, with a naive entropy extraction method applied to them. It is expected that they will not demonstrate ideal properties of randomness, but 8 of 24 tested samples show promising characteristics regardless of these limitations.

Modulo extractor outputs over gap values dictated by the sequencing software may inherit some underlying bias, but the method consistently outperforms nucleotide encoding methods, with the exception of randomly shuffled, encoded and hashed nucleotide sequences. Passing 32-bit words from our sample files, into SHA3-256 (with or without the addition of a timestamp-based salt), results in the modulo extractor surpassing even hashed and shuffled nucleotide sequences for FIPS 140-2, ent, and TestU01 Rabbit and Alphabits. This demonstrates the viability of our approach as a superior method of utilising useful work (nucleotide sequencing) as a source of entropy for random number generation.

When concatenating a 32-bit modulo-derived sequence and a timestamp, then hashing with SHA3-256, only S21 and S22 show any issues. Specifically, they both report a borderline failure of Rabbit's Multinomial test 1. All other test results, for FIPS 140-2, ent, and TestU01 pass for all modulo extractor derived samples. This further reinforces the benefits of our approach over previously documented methods of utilizing nucleotide sequences as a source of randomness.

Considering the sensitive nature of some nucleotide data, particularly human or patented genetic data, we must consider privacy. Our method, unlike the use of nucleotide sequences, ensures that one cannot reverse engineer genetic data from either raw, hashed, or salted and hashed output streams. Only salted and hashed raw nucleotide RNGs can make this same claim, and then only with a sufficiently random salt. Also, as our method does not utilise useful biological data that will be stored as a matter of record in the same manner as nucleotide sequences, it is not possible to use repositories of genetic data to gain a frontrunning advantage should the source of a nucleotide-based RNG be identified by attackers. This will be further reinforced as our modulo extractor is improved to operate over streamed output from nanopore-based devices instead of raw signal files.

5 CONCLUSION

In this paper we have introduced a novel method of extracting entropy from the process of sequencing nucleotides using commercial ONT devices. We have identified usable entropy in the processes of nanopore-based sequencers, and continue to refine methods of accessing the inherent noise of nucleotide sequencing processes in such devices. We demonstrate that modulo extractors provide superior output compared to previous works [2, 7, 13] prior to hashing. We show that not only does our extractor provide appropriate levels of entropy to feed a SHA2-based RNG, but that the bitstreams generated by the extractor over fast5 files themselves show promisingly robust properties of randomness, despite low level correlations inherited from the means by which a gap is defined in such files.

Extractors that rely on time-lag observations share weaknesses with RNG-F by failing to remove correlations between output data and the underlying structure of nucleotide sequence timing data. We demonstrate that modulo extraction does not share this weakness, and provide initial proofs of randomness for the modulo method of extracting randomness from nanopore-based sequencer timing data. Statistical randomness tests identify that post-processing of modulo extracted sequences derived from fast5 files is required, but that the modulo extraction of bitstreams from gap data doesn't exhibit egregious bias or correlations for the majority of tested samples. A simple salt and hash method of post-processing eliminates observed bias in all raw modulo outputs.

We identify that base-call data, expressing specific DNA or RNA sequences as strings of chars, has too much inherent structure to be used as a stand-alone source of randomness. Even after a mixing multiple sources, base-pair data remains highly correlated and biased. Hashing, especially if the data is salted with high entropy sequences from another source, can improve the quality of this data. However, this doesn't resolve the underlying bias - it only complicates observation, an issue overcome by more robust tests of randomness. Modulo extraction of randomness from temporal data (gaps) expressed in fast5 files does not share this weakness, but a much larger dataset is required to identify the actual entropy inherent in such systems, the degree of variance between individual biological samples, and to identify methods that more generally harness to inherent entropy of a wider range of nucleotide sequencers. Our current work provides a viable proof of concept, but requires further analysis to demonstrate that entropy produced in this manner is of use to cryptographers and systems experts.

Sequences produced by our method cannot be used to derive any meaningful nucleotide data, as we rely only on the temporal data from a given sequencing operation. Such data has no relationship with nucleotides or any other identifying data.

5.1 Future work

Our research demonstrates the feasibility of entropy extraction from genetic sequencers. However, the Gaps extractor is currently limited by its relatively low throughput in relation to the input size. The high compression rate between nucleotide sequencer timing outputs and the final bitstream is significant, approximately $10^6 : 1$ from raw signal to final bitstream. This limitation arises from the necessity to remove reads entirely to mitigate the highly correlated nature of processed fast5 and fastq pore occupancy data. The need to prune files to prevent initialisation biases, further decreases signal utilization.

Building on the promising results provided by the Gaps methodology, future work will explore whether additional sources of entropy can be extracted directly from sequencing hardware. One promising area of development involves ASIC noise as observed in raw signal files. These advancements will enable us to produce sufficient output to conduct more sophisticated tests, such as NIST SP800-22 and SP800-90B. Our ongoing and future work will focus on implementing a more efficient version of our sequencer-based entropy extractor, subjecting it to the same rigorous standards applied in our review of DNA-based RNG literature.

ACKNOWLEDGMENTS

This work has been funded by Veiovia Ltd., a University of York spin-out company, operating with investment from Oxford Nanopore Technologies Plc (ONT). Darren Hurley-Smith (DHS) has worked as an independent consultant for Veiovia Limited, to develop and validate methods of extracting entropy from nucleotide sequencing hardware. DHS, Remy Lyon (RL), and Roxana I. Teodor (RIT) were hosted by University of York as Visiting Associates of Prof. Ian Bancroft's group in the Centre for Novel Agricultural Products, Department of Biology. The authors would like to thank Dr. Stuart Reid of ONT for helpful advice on the biophysical processes involved in nanopore sequencing, the properties of nanopore data and review of the manuscript. RL and RIT gratefully acknowledge Dr. James Peter Willcocks of ONT for his input on publication strategy and review of the manuscript. The authors would also like to thank Dr. Saqib Kakvi (ISG, Royal Holloway) for their independent critical review of the manuscript.

REFERENCES

[1] Elena Almaraz Luengo, Bittor Alaña Olivares, Luis Javier García Villalba, Julio Hernandez-Castro, and Darren Hurley-Smith. 2023. StringENT test suite: ENT battery revisited for efficient P value computation. *Journal of Cryptographic Engineering* (2023), 1–15.

[2] Dumitru Balanici, Vlad Tomsa, Monica Borda, and Raul Malutan. 2015. Full duplex OTP cryptosystem based on DNA Key for text transmissions. In *Innovative Security Solutions for Information Technology and Communications: 8th International Conference, SECITC 2015, Bucharest, Romania, June 11-12, 2015. Revised Selected Papers 8*. Springer, 39–48.

[3] Yuval Benjamini and Terence P. Speed. 2012. Summarizing and correcting the GC content bias in high-throughput sequencing. *Nucleic Acids Research* 40, 10 (02 2012), e72–e72. https://doi.org/10.1093/nar/gks001 arXiv:https://academic.oup.com/nar/article-pdf/40/10/e72/25335311/gks001.pdf

[4] Robert G Brown, Dirk Eddelbuettel, and David Bauer. 2013. Dieharder: A random number test suite. *Open Source software library, under development* (2013).

[5] Yevgeniy Dodis, David Pointcheval, Sylvain Ruhault, Damien Vergniaud, and Daniel Wichs. 2013. Security analysis of pseudo-random number generators with input: /dev/random is not robust. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 647–658.

[6] N Galtier, G Piganeau, D Mouchiroud, and L Duret. 2001. GC-Content Evolution in Mammalian Genomes: The Biased Gene Conversion Hypothesis. *Genetics* 159, 2 (10 2001), 907–911. https://doi.org/10.1093/genetics/159.2.907 arXiv:https://academic.oup.com/genetics/article-pdf/159/2/907/42035117/genetics0907.pdf

[7] Christy M Gearheart, Benjamin Arazi, and Eric C Rouchka. 2010. DNA-based random number generation in security circuitry. *Biosystems* 100, 3 (2010), 208–214.

[8] Ashish Gehani, Thomas LaBean, and John Reif. 2004. DNA-based cryptography. *Aspects of molecular computing: essays dedicated to tom head, on the occasion of his 70th birthday* (2004), 167–188.

[9] Zvi Gutterman, Benny Pinkas, and Tzachy Reinman. 2006. Analysis of the linux random number generator. In *2006 IEEE Symposium on Security and Privacy (S&P'06)*. IEEE, 15–pp.

[10] Darren Hurley-Smith and Julio Hernandez-Castro. 2020. Quantum leap and crash: Searching and finding bias in quantum random number generators. *ACM Transactions on Privacy and Security (TOPS)* 23, 3 (2020), 1–25.

[11] Darren Hurley-Smith, Constantinos Patsakis, and Julio Hernandez-Castro. 2020. On the unbearable lightness of FIPS 140–2 randomness tests. *IEEE Transactions on Information Forensics and Security* 17 (2020), 3946–3958.

[12] Pierre L'ecuyer and Richard Simard. 2007. TestU01: AC library for empirical testing of random number generators. *ACM Transactions on Mathematical Software (TOMS)* 33, 4 (2007), 1–40.

[13] Xin-she Li, Lei Zhang, and Yu-pu Hu. 2008. A novel generation key scheme based on DNA. In *2008 International Conference on Computational Intelligence and Security*, Vol. 1. IEEE, 264–266.

[14] Hengyun Lu, Francesca Giordano, and Zemin Ning. 2016. Oxford Nanopore MinION sequencing and genome assembly. *Genomics, proteomics & bioinformatics* 14, 5 (2016), 265–279.

[15] Christian Matt and Ueli Maurer. 2013. The one-time pad revisited. In *2013 IEEE International Symposium on Information Theory*. IEEE, 2706–2710.

[16] Craig G Nevill-Manning and Ian H Witten. 1999. Protein is incompressible. In *Proceedings DCC'99 Data Compression Conference (Cat. No. PR00096)*. IEEE, 257–266.

[17] Kyung Hwan Park, Seong Mo Park, Byoung Gun Choi, Jong Bum Kim, and Kwang Jae Son. 2020. High rate true random number generator using beta radiation. In *AIP Conference Proceedings*, Vol. 2295. AIP Publishing LLC, 020020.

[18] Pramod Pavithran, Sheena Mathew, Suyel Namasudra, and Ashish Singh. 2023. Enhancing randomness of the ciphertext generated by DNA-based cryptosystem and finite state machine. *Cluster Computing* 26, 2 (2023), 1035–1051.

[19] Andrew Rukhin, Juan Soto, and James Nechvatal. 2010. A statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST DTIC Document. *NIST SP800-22* (2010).

[20] Marek Życzkowski, Grzegorz Kwaśnik, Mieczysław Szustakowski, Mateusz Karol, Łukasz Olszewski, and Konrad Dominik Brewczyński. 2017. Encryption key generator based on passive optical elements. In *Optical Fibers and Their Applications 2017*, Vol. 10325. SPIE, 210–215.