Cryptography and Randomness Extraction in the Multi-Stream Model

Dissertation Submitted to

Tsinghua University

in partial fulfillment of the requirement

for the degree of

Doctor of Philosophy

in

Computer Science and Technology

by

Guang Yang

Dissertation Supervisor: Periklis A. Papakonstantinou

December 2015

Cryptography and Randomness Extraction in the Multi-Stream Model

by

Guang Yang

Submitted to the Institute for Interdisciplinary Information Sciences
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at

TSINGHUA UNIVERSITY

December 2015

Author .....
Institute for Interdisciplinary Information Sciences
December 20, 2015

Certified by .....
Periklis A. Papakonstantinou
Assistant Professor
Thesis Supervisor

Cryptography and Randomness Extraction in the Multi-Stream Model

by

Guang Yang

Submitted to the Tsinghua University
on December 20, 2015, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

We initiate and develop from the ground up cryptography and randomness extraction over Very Big Data Objects. These big objects are handled in the multi-stream model, for which we study constructions as well as limitations. We propose new views and concepts, devise new constructions and methods, introduce new lower bound techniques, and perform extensive experimental evaluations.

The multi-stream model captures the essence of computation over external storage, with severely restricted local memory and sequentially making only few passes over the storage. Typically, we consider $\mathrm{O}(\log n)$ memory size and constant many read/write *streams*, where each stream is a big tape that can be scanned sequentially from left to right for only very few times, i.e. constant or slightly above that.

For cryptography over Very Big Data Objects, our study focuses on the theoretical foundations. Based on mild intractability assumptions, we construct one-way functions and pseudorandom generators within constant many passes. From the Learning With Errors (LWE) assumption, we devise a Public-Key Encryption (PKE) scheme where both the encryption and decryption use constant many passes. We also complement the constructions with impossibility of constant-pass super-linear stretch pseudorandom generators and a linear lower bound for the length of private-keys in PKE. We prove this using a new lower bound technique that we develop.

For randomness extraction over Very Big Data Objects, we obtain a streaming randomness extractor that makes $\mathrm{O}\left(\log\log\frac{n}{\varepsilon}\right)$ passes, which is only slightly above constant. Interestingly, we show that this bound is tight for every randomness extractor in our streaming model. In addition to the theoretical developments, we realize and empirically validate our extractor as an ultra-efficient executable program, which is used to extract randomness from Big Data. This is a novel conceptual view of real-world data as big sources of entropy. Both in theory and in practice, this is the first method able to extract high-quality random bits from big objects.

Dissertation Supervisor: Assistant Professor Periklis A. Papakonstantinou

Acknowledgments

First of all, I should thank my parents and my girlfriend, Fei Tian, for their support and encouragement to pursue an academic career. Then, my sincerest gratitude goes to my advisor and friend Prof. Periklis A. Papakonstantinou, whose supervision was indispensable to this program. Periklis has high research standards and offered me guidance and support in all aspects. Let me also thank Prof. Andrew C. Yao, Dean of Institute for Interdisciplinary Information Sciences here at Tsinghua, for creating a research environment I very much enjoyed.

I am grateful to my coauthors: Benny Applebaum, Sergei Artemenko, Jing He, Yuval Ishai, Hongyu Liang, Ronen Shaltiel, Mor Weiss, David P. Woodruff, Jia Xu. They devoted countless hours discussing and collaborating with me. I should also express my appreciation to all my colleagues, especially John P. Steinberger, for enlightening discussions and comments on this thesis and beyond.

Finally, many thanks to Andrej Bogdanov, Yuval Ishai, Eyal Kushilevitz, Claudio Orlandi, Omer Reingold, Salil Vadan, and Yu Yu for their invaluable feedback, suggestions, and comments.

This work was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, and the National Natural Science Foundation of China Grant 61033001, 61350110536, 61361136003.

Contents

List of Figures

List of Tables

List of Definitions

Chapter 1

Introduction

Due to the advance in technology from smart phones to wearable devices, and more importantly the spread of Internet, the world is experiencing an explosion of information. A spectacular and ever-increasing amount of data, commonly termed as *Big Data*, are being generated all over the world, which in 2011 already exceeded 5 exabytes (i.e. 5×10^{18} bytes) per day [WK13]. Big Data consists of a variety of unstructured data. These data are produced or collected by individuals, companies and research laboratories, ranging from social networks feeds to DNA sequenced data. At the time of the writing of this dissertation, the study on how to make use of Big Data has attracted a lot of research efforts in theory and in practice.

To theoretically analyze the complexity of algorithms on Big Data, a suitable computation model must be specified. This model should capture the properties and limitations of computation over Big Data, where the major difficulty is that Big Data cannot be stored and processed in the local memory. The standard approach is to distinguish *external storage* from *local memory*, and put restrictions on queries to external storage. In this dissertation, we consider the *multi-stream model*, where the local memory is severely restricted in size, and the external storage is abstracted as *streams*. Every stream is a long tape that provides sequential access from left to right for a small number of times, i.e. very few *passes*. The complexity of an algorithm in the multi-stream model is traditionally measured by three parameters: the size of local memory, the number of streams, and the number of passes over all these

streams.

This dissertation initiates the study of huge size pseudorandom and random objects in the multi-stream model. We construct pseudorandom objects such as *one-way functions*, *pseudorandom generators*, and *public-key encryption schemes*. These are also important *cryptographic primitives*, which shows the feasibility of cryptography in this model. Pseudorandomness, which means computational unpredictable, is the computational analog of true randomness in the statistical sense. Besides the computational view of randomness, the second main theme of this research regards true randomness. In particular, our starting point is that real-world Big Data are sources of low quality (i.e. far from uniform) statistical randomness. Based on this view, we propose a *randomness extractor* in the multi-stream model and devise a complete method to extract randomness from Big Data. Furthermore, we implement this method as an ultra-efficient executable program, and empirically validate its output quality on real-world data with standard uniformity tests [RSN+10, die08]. Regarding limitations, we prove lower bounds for cryptographic primitives and randomness extractors in the multi-stream model, with a new lower bound proof technique inspired by [BH12]. Overall we have conducted a comprehensive and in-depth research on both theoretical and practical aspects.

Figure 1-1 depicts the relation between the main components of this dissertation and previous works. The discussion about the multi-stream model is in Section 1.1. Results and their relation to previous works is summarized in Section 1.2 and 1.3.

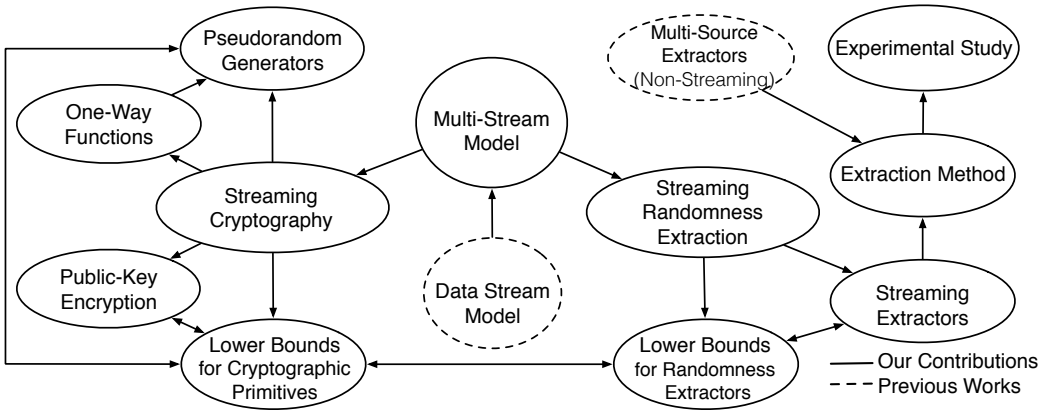

Figure 1-1: **An overview of our contributions.** This work includes a variety of constructions and lower bounds, as well as empirical studies on extraction methods.

1.1 Modeling the Computation over Big Data

In our work, we choose the *multi-stream model*, also known as *multiple read/write stream model* in [BH12], to characterize the computation over Big Data stored in external storage. There are several other alternatives in the literature, such as *space-bounded model*, *parallel disk model* [Vit01], *external memory Turing machines* [AM99], *data stream model* [BBD+02, Mut03], and *reversal-bounded computation model* [CY91, HS08]. However, none of the above is appropriate for our purposes, where we consider large input size (i.e. at least a few GBs that cannot be easily stored in the local memory) together with large output size (of random and pseudorandom objects). Here we briefly introduce these models and explain why we use streams for external storage, and why multiple read/write streams are necessary. The multi-stream model is discussed in Section 1.1.1.

Why streaming? The space-bounded model, which is defined through Turing machines with bounded memory plus read-only yet random[1] accessible input, only captures half the essence of computation over big objects. The classical parallel disk model is still rough by simply measuring the amount of information read from external storage. The *external memory Turing machine* has a single external tape with random access, and allows arbitrary computation with unbounded working memory between any two Input/Output (IO) operations on the external tape. These models have achieved various theoretical success in characterizing computation over external storage. However, by allowing random access to external storage, these models ignore an important nature in practice – smart heuristics are applied in modern computers to minimize the overhead of operations on external storage, and these heuristics prefer sequential access to random access. In this sense, we believe it is more appropriate to model external storage in a streaming fashion, which captures the computation of various statistics on data streams such as network monitoring, transactional traffic,

[1] "Random access" means that every element in a set can be accessed with the same cost, regardless of how many elements are there or how these elements are aligned or structured. Although literally containing "random", the term "random access" has nothing to do with random bits or randomness discussed in this dissertation.

and database systems.

Why multiple streams? The *data stream model* [BBD+02, Mut03] has achieved prominent success in both theoretical and practical studies. In this model, the local memory size is bounded and the input is given as a *stream*, which is a unbounded tape that must be scanned in one direction, e.g. from left to right. The number of sequential passes over the stream is a crucial parameter. Ideally we wish to spend a single *pass* over the input stream. Nevertheless, when realizing the input stream as an external storage device that is not read-once, it also makes sense to consider p passes for a small integer p, when p is constant or slightly above constant.

The limitations of the data stream model are also theoretically studied, and various lower bound are well established. In particular, no pseudorandom objects can be computed in this single-stream model using local memory size $\mathrm{O}(\log n)$ and constant many passes (see Theorem 2.5 on page 28, improved on [KGY89]). Thus, it becomes imperative to investigate the model with multiple streams.[2]

1.1.1 The Multi-Stream Model

It is observed [GS05] that for many natural problems the large external storage can be abstracted by multiple read/write streams rather than a single one. For example, when realizing every data stream with a hard disk drive, it is quite straightforward to think of using more than one stream in the model describing computation with multiple hard disk drives. By adding more streams to the data stream model, the multi-stream model captures the additional ability of external storage and is further studied in [BH12, BJR07, GHS09].

Formally, the *multi-stream model* is based on a (p, s, t)-*bounded Turing machine*, where $p = p(n), s = s(n)$, and $t = t(n)$ are functions of input length n.

Definition 1.1. *A* (p, s, t)-bounded Turing machine *is a standard multi-tape Turing machine with* $(t + 1)$*-many tapes. The first tape represents the local memory and its*

[2]The *reversal complexity model* [CY91, HS08] is similar but unnecessary for our purposes, though it is slight stronger with two-directional accesses.

size is bounded by s. The other t tapes are unbounded in size but allow only p many unidirectional sequential passes (i.e. scanning from left to right) over them. These t tapes are called streams*, and the first stream is always viewed as the (read/write) input tape. The passes over different streams do not have to be synchronized.*

A (p, s, t)-streaming algorithm *is an algorithm computable with a (p, s, t)-bounded Turing machine. A function is* (p, s, t)-streaming computable *if it can be evaluated by a (p, s, t)-streaming algorithm.*

Typically, we consider $p = \mathrm{o}(\log n)$, $s = \mathrm{O}(\log n)$, and constant t. Unless otherwise specified, the previous typical values are assumed whenever (p, s, t) is omitted, and "log" denotes the logarithm to base 2.

Example. Figure 1-2 depicts an example of the streaming algorithm that adds two binary[3] integers 11100 and 110. The algorithm first copies 110 to the second stream, with two passes over both streams; then, by scanning two streams simultaneously, it adds up 11100 and 110 without carries; finally, it scans the first stream to deal with carries, while writing to the second stream. In the last step, the algorithm maintains a counter in its local memory to count how many tape cells on the first stream have been scanned before seeing the next "0" or "2". Whenever reaching such a digit, it writes $100\cdots0$ if reads "2" or $011\cdots1$ if reads "0", where the length of $100\cdots0$ (resp. $011\cdots1$) equals to the value stored in the counter.

The multi-stream model is substantially stronger than the data stream model. For example, a list of n integer elements $\{1, 2, \ldots, n\}$ can be sorted with $s = \mathrm{O}(\log n)$ local memory and $p = \mathrm{O}(\log n)$ passes over 3 streams; whereas with a single read/write stream, sorting n elements requires $sp = \Omega(n)$ by a standard *communication complexity* argument [Yao79], since in average-cases $\Omega(n)$ many elements must be moved from the first half of the input to the second half. In this dissertation, by enhancing the data stream model with a second read/write stream, we manage to circumvent a

[3]The alphabet in the description is $\{0, 1, 2, +\}$, which can be encoded with 2 bits and with two passes over two streams it is trivial to convert between strings with elements from this alphabet and the binary representation with elements $\{0, 1\}$. For example, encode "0" with 00 and "1" with 01, then it is easy to convert "110" (encoded by 010100) to 110.

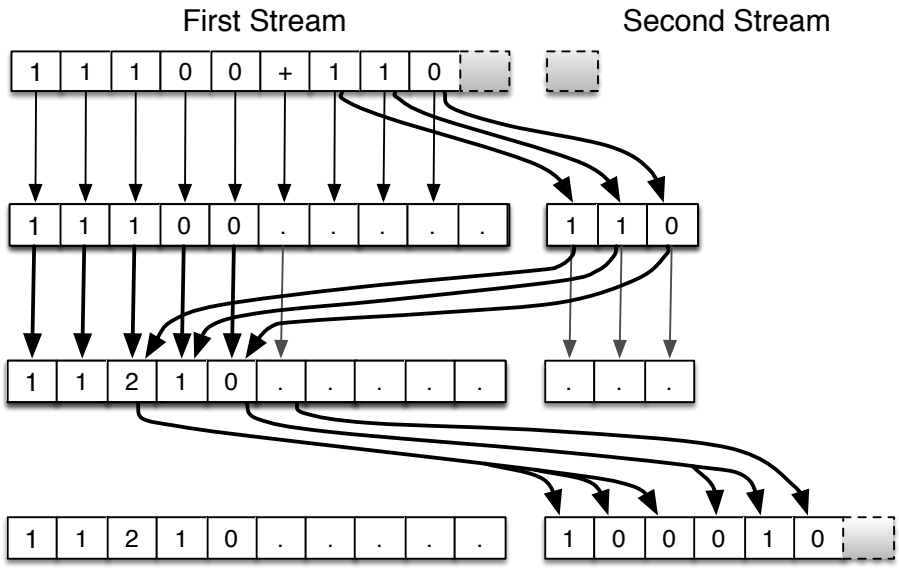

Figure 1-2: **A streaming algorithm example for adding two binary integers.**

number of previous limitations [BYRST02, KGY89, Pap10, PY12] and thus we make cryptography and randomness extraction feasible.

While increasing feasibility, the stronger model relaxes limitations that previous lower bound techniques are based on. As a result, there are very few lower bound results in the multi-stream model. To the best of our knowledge, except for $\omega(\log n)$ lower bounds [CY91, Pap94] based on space hierarchy theorem and diagonalization arguments, the first logarithmic lower bound is introduced in [GS05]. It shows that $\Omega(\log n)$ many passes are necessary for sorting n elements with $O(\log n)$ local memory and constant many streams. Then, [BH12] proves another logarithmic lower bound in the number of passes for approximating frequency moments. Both works indicate that with sub-logarithmic number of passes, the multi-stream model does not attain significant advantage over the data stream model.

This multi-stream model with reasonable parameters is exactly what we consider in the dissertation. Unless otherwise specified, a *streaming algorithm* means an algorithm computable with $O(\log n)$ local memory and $o(\log n)$ passes over 2 streams. In fact, our constructions use constant or at most $O(\log \log n)$ passes in total. For lower bounds, with the new proof technique inspired by and significantly extend the machinery of [BH12] we have obtained lower bounds for *families of functions*. In other

words, we do not have a concrete function to discuss, which is the case in all known lower bounds people proved before our work. The argument information-theoretically makes use of the minimum property shared by all functions from the whole family.

We strictly limit the number of passes for both theoretical and practical interest. In theory, only a sub-logarithmic number of passes is interesting because following [CY91, HS08], an algorithm using $\mathrm{O}(\log n)$ local memory and $\mathrm{O}(\log n)$ passes over 2 streams suffices for every function in Log-space, where Log-space is the standard space-bounded complexity class that allows $\mathrm{O}(\log n)$ local memory and unrestricted random access to the read-only input. More concretely, we can double the number of input copies using 3 passes over 2 streams, by scanning and copying the input twice to the second stream. Therefore, with $\mathrm{O}(\log n)$ many iterations we obtain $\mathsf{poly}(n)$ many copies of the input, and hence every Log-space function can be evaluated by with a single scan through those copies. Although $\mathrm{O}(\log n)$ many passes is not extremely prohibitive in practice, the polynomial blow-up in stream length detaches the model from practice by ruining the correspondence between streams and external storage. Streams are introduced to characterize the computation on input of huge size, and hence another polynomial blow-up in size is intolerable. In fact, the worst-case upper bound of the space used on the streams is exponentially related to the number of passes [GS05].

1.2 Cryptography in the Multi-Stream Model

We begin with the motivation of obtaining cryptographic primitives in the multi-stream model in Section 1.2.1. Then, we informally introduce cryptographic primitives in Section 1.2.2 and exhibit our constructions and lower bound results in Section 1.2.3. In Section 1.2.4 we summarize related previous works.

1.2.1 Motivations

In modern cryptography, the security commonly refers to computational security against adversaries with limited computing power. This kind of security is described

with a function $s(n)$ in the security parameter n, where n is usually the key length, such that any adversary must spend $s(n)$ times more computing power than an authorized user who holds the key to solve the same problem. Most cryptographic constructions consider intractable problems with only mild security parameters that are smaller than the local memory size, i.e. the keys reside in the local memory and allow unlimited access. However, such mild parameters may not induce a satisfiable security $s(n)$, especially when the underlying problems are "not so hard", e.g. the *approximate GCD (Greatest Common Divisor) problem*[4]. Therefore, it is natural and theoretically motivated to question the necessity of mild key length – since computational security is described in an asymptotic way, why the security parameter must be bounded by the local memory size? Is it possible to achieve cryptographically secure functions for very long keys?

Formally, we put forth the question of whether cryptography is feasible in the multi-stream model, where the local memory is smaller than the security parameter, against arbitrary polynomial time adversaries, who are as powerful as usual. This question is motivated in practical settings where the keys and messages are huge in size, and in theory it falls in the fundamental research of cryptography with rudimentary resources.

1.2.2 Cryptographic Primitives

We consider private-key and public-key cryptographic primitives: one-way functions, pseudorandom generators, and public-key encryption (PKE) schemes. In the following, we briefly explain what these primitives are, while the formal definitions are given in Section 2.1 on page 25.

A *one-way function* is a function that is easy to compute (i.e. in polynomial running time) but hard to invert. For example, multiplying two large integers is considerably easier than factoring the product of two unknown large (prime) integers,

[4]The approximate GCD problem asks to find a common divisor p from a bunch of near multiples of p, i.e. $\{pq_i + r_i \mid p, q_i, r_i \text{ are integers and } |r_i| \ll p \text{ for } i = 1, 2, \ldots, m\}$ for an integer $m \geq 2$. This problem is related to factoring RSA modulus with partial information [HG01].

since *Integer Factorization* is one of the standard intractability assumptions and no polynomial time algorithm is known so far.

A *pseudorandom generator* is a polynomial time deterministic algorithm that stretches n-bit random to $\ell(n) > n$ pseudorandom bits, where *pseudorandom* means indistinguishable from random bits by any polynomial time distinguisher.

A *PKE scheme* consists of three polynomial time algorithms for key-generation, encryption and decryption respectively. The scheme first takes the security parameter as input and generates a public encryption key and a private decryption key. These two different (asymmetric) keys are used for encryption and decryption respectively. A PKE scheme achieves *Indistinguishability under Chosen Plaintext Attack* (IND-CPA) security (also known as *semantic security*) if for any two messages, even chosen by an adversary, the encrypted ciphertext cannot be distinguished.

Note that unconditional constructions of the above cryptographic primitives are impossible under current knowledge. In fact, any such construction will immediately imply $\mathsf{P} \neq \mathsf{NP}$, which has been the most fundamental open problem in theoretical computer science for decades. To bypass this impossibility, all cryptographic primitives are constructed based on *intractability assumptions* (also known as *cryptographic hardness assumptions*). For example, Integer Factorization, which asks to decompose a large composite integer into smaller integers, is a standard and popular intractability assumption.

1.2.3 Our Results

We first exhibit our constructions and then complement them with lower bounds.

Possibility results. We achieve the following constructions for private-key primitives, which are formally discussed in Section 2.3 and 2.4. These constructions are based on the general assumption that one-way functions exist in NC^1, where NC^1 is the class of functions computable with $\mathrm{O}(\log n)$-depth circuits consisting of $\mathsf{poly}(n)$-many constant fan-in gates; see Section 2.1 on page 24 for a precise definition, and the textbook [AB09] for more details. This assumption is quite mild since NC^1 is suf-

fices for many standard intractability assumptions , including Integer Factorization, Decoding Random Linear Codes, the Subset-Sum Problem (see the textbook [Gol01]) and Learning With Errors ([Reg05]).

Theorem 2.8 (informally stated). If one-way functions exist in NC^1, then there is a $(5, \mathrm{O}(\log n), 2)$-streaming computable one-way function, i.e. computable with 5 passes, $\mathrm{O}(\log n)$ local memory, and 2 streams.

Theorem 2.12 (informally stated). If one-way functions exist in NC^1, then there exist pseudorandom generators that are $(7, \mathrm{O}(\log n), 2)$-streaming computable.

For public-key cryptography, we have the following PKE scheme based on the standard *Learning With Errors* (LWE) assumption (the decisional version [Reg05], see Assumption 2.18 on page 45 for definition). The construction is presented and analyzed in Section 2.5. Note that the key generation procedure in our construction is not implemented in a streaming fashion.[5]

Theorem 2.16 (informally stated). If the decision-LWE assumption holds, there is a PKE scheme with both encryption and decryption $(\mathrm{O}(1), \mathrm{O}(\log n), 2)$-streaming computable.

Impossibility results. We introduce a new proof technique (inspired by [BH12], see Section 4 for details) for lower bounds of random and pseudorandom objects in the multi-stream model. With this technique, we show the lower bound in passes for super-linear stretch pseudorandom generators in the multi-stream model.

Theorem 4.5 (informally stated). No pseudorandom generator with super-linear stretch $\ell(n) = \omega(n)$ is computable with $\mathrm{O}(\log n)$ memory and constant many passes.

The same technique is used to prove a lower bound for PKE.

Theorem 4.8 (informally stated). For every IND-CPA secure (defined on page 26) PKE scheme whose decryption is a streaming algorithm within constant many passes,

[5]This drawback is acceptable since the key generation is only invoked once in the offline phase for multiple runs of (online) encryption/decryption. In fact, the key generation in our construction involves matrix multiplication which are inherently not streaming computable.

the length of its private decryption key must be at least linear in the length of the plaintext.

1.2.4 Comparison to Previous Works

Our constructions have theoretical merit by showing the feasibility of cryptography in the multi-stream model, which is unexpected. First, it is unexpected because single-stream one-way functions do not exist, i.e. no $(\mathrm{O}(1), \mathrm{O}(\log n), 1)$-streaming algorithm can compute any one-way function; see Theorem 2.5 (page 28) where we show this (also improves on [KGY89]). Also, black-box constructions of cryptographic primitives in the multi-stream model are indeed vastly impossible, even from concrete intractability assumptions. For example, any black-box use of lattice assumptions, e.g. the *Decoding Random Linear Codes* (DRLC) assumption (see Assumption 2.21 on pager 53 for definition), must intrinsically rely on multiplying a matrix by a vector inside the black-box, which needs $\Omega(\log n)$ many passes by [GHS09]. We note that [BJP11] ruled out families of black-box constructions and they conjectured impossibility of streaming cryptography with a constant number of passes. We refute that conjecture with explicit constructions in this dissertation.

Of course, we cannot beat lower bounds. We bypass the single-stream lower bound by adding the second stream, and we circumvent the limitation of the multi-stream model by resorting to non-black-box constructions. This way we bring the number of passes down to a constant and at the same time base our constructions on general assumptions, which contrasts folk wisdom[6] in streaming computation.

More specifically, we take *randomized encodings* of the NC^1 computation of the assumed one-way function, and thus obtain an information-theoretically equivalent function that is computable in the multi-stream model. The randomized encoding technique (see Section 2.1 on page 26 for details) was introduced by the seminal work [AIK06b] and its followups [AIK06a, AIK08] for cryptographic constructions in NC^0,

[6]It was believed that for common types of functions, if when adding a second tape helps then permuting the input in the single-stream model will help as well. But a permuted one-way function is still one-way.

where NC^0 (defined on page 24) is technically orthogonal to the multi-stream model. These are non-black-box constructions since the computation itself is encoded and hence the constructions rely on the knowledge of how to compute those functions.

For a theoretical discussion on the existence of one-way functions and pseudorandom generators in the multi-stream model, the assumption can be relaxed to existence of one-way functions in Log-space (see Section 2.1 on page 24). Figure 1-3 depicts a circle of implications for one-way functions/pseudorandom generators. If there is an one-way function in any of the four computational complexity classes, then both one-way functions and pseudorandom generators exist in all of them. Note that pseudorandom generators are in particular one-way functions, and however, nothing similar is known for PKE.

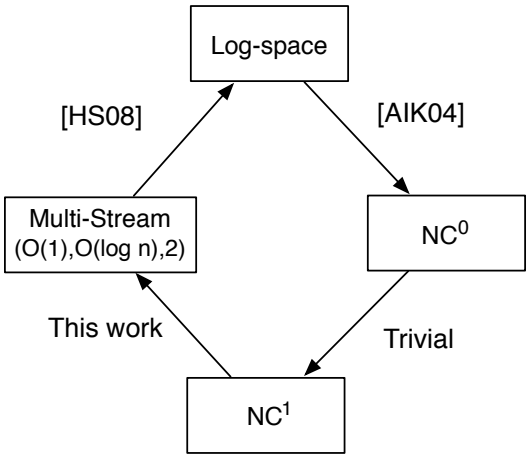

Figure 1-3: **Implications for the existence of one-way functions/pseudorandom generators.**

1.3 Randomness Extraction in the Multi-Stream Model

We also consider randomness extraction in the multi-stream model, which is not only of theoretical interests but also leads to the empirical method of extracting randomness from Big Data. We first discuss the motivation, the formal context, then

present our conceptional, theoretical and experimental contributions.

1.3.1 What is Randomness Extraction?

Why randomness extraction? True randomness is a valuable resource and indispensable for multiple disciplines ranging from cryptography and game theory to algorithm design and numerical simulation of physical systems. However, obtaining high quality random bits, i.e. uniform or statistically close to uniform ones, is in general difficult. One major obstacle is that high quality random bits can only be generated from a sequence of uncertain events that contain *true randomness*. At the time of the writing of this dissertation, the ultimate characterization of uncertain events and true randomness remains unclear in mathematics and controversial in philosophy. Therefore, it is natural and legitimated to put aside that obstacle for a moment, assume the existence of uncertain events, and consider what shall happen thereafter.

Even when given recorded uncertain events, it is still a great challenge to obtain high quality random bits, because those events are not perfect random and may be correlated. For example, consider flipping a biased coin many times, or experiments with more involved correlations. Suppose the uncertain events are collected to constitute a random variable X over $\{0,1\}^n$. In most cases X is not necessarily uniformly distributed, and in fact it can be far from uniform. Thus, extracting high quality random bits from such "weakly" random X is a natural demand, which has motivated various theoretical approaches on randomness extraction.

Here we put randomness extraction in formal context.

Definition 1.2. *For every random variable X over $\{0,1\}^n$, the* (Shannon) entropy *of X is denoted as*

$$\mathsf{H}[X] \stackrel{def}{=} \sum_{x \in \{0,1\}^n} \Pr[X = x] \log_2 \left(\Pr[X = x]^{-1} \right)$$

As the worst-case analog of entropy, the min-entropy*[CG88] of X is $\mathsf{H}_\infty[X]$ defined*

as

$$\mathsf{H}_\infty[X] \stackrel{def}{=} \min_{x\in\{0,1\}^n} \log_2\left(\Pr[X=x]^{-1}\right)$$

Definition 1.3. *The* min-entropy rate *of X is denoted by $\kappa \stackrel{def}{=} \mathsf{H}_\infty[X]/n$, and X is called a* weak source *if $\kappa < 1$. In particular, we call X an (n,k)-*source *if $\mathsf{H}_\infty[X] \geq k$.*

Unless otherwise specified, all sources that we consider in this dissertation are (n,k)-sources with $k = \Omega(n)$.

We generalize the above (Shannon) entropy and min-entropy to the conditional form. Note that conditional min-entropy takes a worst-case quantity on conditions.

Definition 1.4. *For a random variable Y taking values over S, (i) the* conditional (Shannon) entropy *of X conditioned on Y is defined as $\mathsf{H}[X\big|Y] \stackrel{def}{=} \sum_{y\in S}\Pr[Y=y]\mathsf{H}[X\big|Y=y]$; and (ii) the* conditional min-entropy *of X conditioned on Y is defined as $\mathsf{H}_\infty[X\big|Y] \stackrel{def}{=} \min_{y\in S}\mathsf{H}_\infty[X\big|Y=y]$.*
Let $X=(X_1,\ldots,X_m)$ be the concatenation of m random variables. We say that (i) every block of X has next-block-entropy *k if for every $i\in[m]$, $\mathsf{H}[X_i\big|X_1,\ldots,X_{i-1}] \geq k$; and as the worst-case analog, (ii) every block of X has* next-block-min-entropy *k if for every $i\in[m]$, $\mathsf{H}_\infty[X_i\big|X_1,\ldots,X_{i-1}] \geq k$.*

Definition 1.5. *A* randomness extractor *$\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a function that takes two inputs: one is sampled from the source X over $\{0,1\}^n$, and the other is a d-bit random seed $Y \sim \mathcal{U}_d$. Fixing an* error tolerate *$\varepsilon > 0$, we say that Ext is a (k,ε)-*extractor*, if for every (n,k)-source X, the output distribution $\mathsf{Ext}(X,Y)$ is ε-*close to the uniform distribution*, i.e. $\frac{1}{2}\sum_{z\in\{0,1\}^m}|\Pr[\mathsf{Ext}(X,Y)=z] - \frac{1}{2^m}| \leq \varepsilon$.*

We discuss randomness extractors in the most practically relevant context and always require $d < m$. In typical settings, the seed length is usually $d = \mathsf{polylog}(n) \stackrel{\text{def}}{=} (\log_2 n)^c$ for a constant $c > 0$. The seed is necessary since otherwise it is impossible to extract a single random bit from one weak source [Sha04].

The are several variants of randomness extractors in the literature. For example, *multi-source extractors* [CG88, DEOR04, Rao09a, Raz05] work on multiple independent sources and do not have to use a random seed. The second (and last) important

variant of extractor works only for specific sources, e.g. *bit-fixing* sources (some bits are fixed and the rest bits uniformly distributed) or affine sources (all the supports constitute an affine subspace), rather than any (n, k)-source. These variants are also considered in our work (Chapter 3).

1.3.2 Extraction from Big Data – Conceptual Contributions

As discussed in Section 1.3.1, the generation of high quality random bits must rely on a source of randomness, i.e. a sequence of uncertain events. In this section we bring up the idea of viewing Big Data as a weak source and evaluate the gains and obstacles. We put randomness extraction from Big Data in the appropriate rigorous context in Section 1.3.3.

The world bustles with uncertain events whose outcomes are routinely recorded as Big Data. These include data generated in laboratories (e.g. DNA sequenced data) and data produced by individuals (e.g. social network feeds). We view the generation of such data as the sampling process from a *big source*, which is a random variable of size at least a few GBs. This view initiates the study of big sources in true randomness extraction.

Two competing factors arise in big source extraction. First, input samples must be locally processed due to their size; second, all statistical dependencies embedded arbitrarily in input samples must be removed. *Ad hoc* solutions bypass this issue by splitting a big sample into smaller ones and processing every smaller sample individually, and thus they rely on independence assumptions that restrict applicability. In this dissertation, we consider big source extraction in the multi-stream model that locally process everything, and we also make mathematical analysis to prove the extraction quality from any weak sources.

There are tangible benefits to linking randomness to Big Data. First, they are everywhere [Ban14, WK13]. Second, these data are in common use, so adversaries cannot significantly reduce statistical entropy without making them unusable [CMBH02]. In addition, these data are accessible without any special equipment, in contrast to previous works those resort to physical phenomena such as quantum experiments

[PAM+10, PM13, UZZ+13, VV12] or atmospheric turbulence [MVV14]. The ability to extract from big samples also leverages the study of quantum randomness expanders, since it allows us to post-process while ignoring local statistical dependencies.

1.3.3 Extraction from Big Data – Technical Contributions

We consider randomness extraction in the multi-stream model (defined in Section 1.1.1) with $(\mathrm{O}(\log\log n), \mathsf{polylog}(n), 2)$-bounded Turing machines, i.e. using $\mathsf{polylog}(n)$ local memory and $\mathrm{O}(\log\log n)$ passes over 2 streams. The weak source X is initially stored on the first stream, while the other stream is auxiliary. The seed permanently resides in the local memory of size $\mathsf{polylog}(n)$.

In this part, we first exposit the lower bound results, and then we show that our lower bound in some sense is tight by presenting a matching construction.

Lower bounds. We show that $\Omega\left(\log\log\frac{1}{\varepsilon}\right)$ many passes are necessary for extractors in the multi-stream model. Furthermore, we prove a much stronger lower bound, i.e. $\Omega(\log n)$ many passes, for *oblivious* (k,ε)-extractors with $k = n^{1-\Omega(1)}$ and $\varepsilon < 1/2$. The lower bound results are obtained using the same technique as mentioned in Section 1.2.3 on page 9. The proofs are given in Section 4.3.

Theorem 4.9 (informally stated). Let $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a fixed (k,ε)-extractor. If Ext uses a seed of length $d \leq \mathrm{O}(s)$ and is (p,s,t)-streaming computable for a positive constant t and $s = n^{\mathrm{o}(1)}$, then the streaming algorithm for Ext must make at least $p = \Omega\left(\log\log\frac{1}{\varepsilon}\right)$ many passes, even for bit-fixing sources.

We say that an extractor is *oblivious* if after fixing the bits of the seed the head-move on the streams depends only on the input length. We prove the following lower bound for oblivious extractors.

Theorem 4.11 (informally stated). Suppose $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is an extractor obliviously computable with a (p,s,t)-bounded Turing machine where t is a constant and $s = n^{o(1)}$. If Ext is a (k,ε)-extractor with $k = n^{0.99}$ and $\varepsilon < 1/2$,

and $d \leq m^{0.99} = k^{\Omega(1)}$, then the streaming algorithm for Ext must make at least $p = \Omega(\log n)$ many passes, even for bit-fixing sources.

Note that all known general extractors (streaming or not), by the time when this dissertation is written, are oblivious.

Constructions. On the positive side, our main contribution is a two-stream seeded extractor, which we call *Random Re-Bucketing* (RRB). RRB is a streaming algorithm that uses local memory of size $\mathsf{polylog}(n)$, a $\mathsf{polylog}(n)$ random seed, and two read/write streams. It makes $\mathrm{O}\left(\log(\log\frac{1}{\varepsilon} + \log n)\right) = \mathrm{O}\left(\log(\log\frac{n}{\varepsilon})\right)$ many passes to produce an output ε-close to the uniform distribution for input sources of min-entropy $k = \Omega(n)$. We obtain the following provable guarantees.

Theorem 3.17 (informally stated). For every $\varepsilon > 0$ and $k = \Omega(n)$, there exists an integer $\lambda = \mathrm{O}\left(\log\frac{n}{\varepsilon}\right)$, $d = \mathrm{O}\left(\lambda \log n \log\frac{n}{\varepsilon}\right)$ and $m = \Omega\left(\lambda n / \log\frac{n}{\varepsilon}\right)$, such that $\mathsf{RRB} : \{0,1\}^{\lambda \times n} \times \{0,1\}^d \to \{0,1\}^m$ is a (k,ε)-extractor for λ many independent (n,k)-sources. Here, the input stream to RRB consists of λ samples of all the independent sources listed one after the other in the same stream.

In the above theorem, the λ-many sources are independent but not necessarily identically distributed. These sources are transparent in the algorithm and only needed for the proof, i.e. RRB treats all their samples as a whole input stream. The same RRB algorithm, with even better parameters, also works on a single sample from specific sources, i.e. bit-fixing or affine sources (Definition 3.1 and 3.2 on page 56).

Theorem 3.5 (informally stated). For every $\varepsilon > 0$ and $k = \Omega(n)$, there exists $d = \mathrm{O}\left(\log n \log\frac{n}{\varepsilon}\right)$ and $m = k - \mathrm{o}(k)$, such that $\mathsf{RRB} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k,ε)-extractor for every (n,k)-bit-fixing source.

Theorem 3.12 (informally stated). For every $\varepsilon > 0$ and $k = \Omega(n)$, there exists $d = \mathrm{O}\left(\log n \log\frac{n}{\varepsilon}\right)$ and $m = \Omega\left(n/(\log\frac{n}{\varepsilon})\right)$, such that $\mathsf{RRB} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k,ε)-extractor for every (n,k)-affine source.

In the most typical case where $\varepsilon = 1/\mathsf{poly}(n)$, both the local memory size and seed length in above results become $\mathrm{O}\left(\log^3 n\right)$ or $\mathrm{O}\left(\log^2 n\right)$, and the number of passes is $\mathrm{O}\left(\log\log n\right)$. RRB is asymptotically optimal in the number of passes, since we have the tight $\Omega\left(\log\log \frac{1}{\varepsilon}\right) = \Omega\left(\log\log n\right)$ lower bound (page 16).

For a comprehensive treatment of parameters, we note that one can beat the $\Omega\left(\log\log \frac{1}{\varepsilon}\right)$ bound in the number of passes by using a *very long* seed. In particular, we can construct a two-stream extractor (Section 2.2 on page 31) that makes a constant number of passes and uses a seed of length n^2 to extract $\Omega\left(k\right) + n^2$ many bits. However, this construction is mainly of theoretical relevance, since almost every practical application calls for exponentially smaller seeds.

1.3.4 Extraction from Big Data – Experimental Contribution

We propose an empirical method for true randomness extraction from Big Data, and implement the method as an executable program. Then, we experimentally evaluate the efficiency and quality of the proposed method on real-world data.

The extraction method consists of a big data randomness extractor and the necessary components for initial seed generation and parameter estimation. The description of our method is given below.

i. RRB is the big data extractor and it is implemented as an ultra-efficient executable program, whose output quality is empirically validated (Table 3.1, Table 3.2, Table 3.4, and Table 3.5 in Section 3.4) by standard statistical tests [RSN+10, die08].

ii. The initial seed generation method consists of two phases. First, apply a multi-source extractor following [BIW06, Zuc90] on multiple independent sources to extract the very initial randomness without using any seed. Then, invoke RRB to expand the very initial randomness further.

iii. The parameter estimation method is used to determine necessary parameters to run the RRB extractor, e.g. it estimates a lower bound for the min-entropy k.

The proposed method is validated in terms of efficiency and quality (measured by two standard quality test suites, NIST [RSN+10] and DIEHARD [die08]) with spectacular results (in terms of the reported measurements) on real-world data samples. These samples range in size from 1.5 GB – 20 GB and they are from 12 data categories (see Table 3.6): compressed/uncompressed text, video, images, audio, DNA sequenced data, and social network data. The empirical extraction is for $\varepsilon = 10^{-20}$ and estimated min-entropy rate ranging from 1/64 to 1/2, with running time from 0.85 hours to 11.06 hours and less than 22 MB of memory on a desktop PC (Figure 3-2-A). Note that prior to this work, processing a 20 GB sample would have taken more than $100,000$ years of computation time and exabytes of memory (Figure 3-2-B). The extracted outputs of our method pass all quality tests in the same way as the ideal uniform (Table 3.1), whereas the before-extraction-datasets fail almost everywhere (Table 3.2). Such test results provide further evidence supporting that the extraction quality is statistically close to the ideal uniform distribution, besides the necessary [SB00] rigorous mathematical treatment.

1.3.5 Comparison to Previous Works

For single-stream algorithms, [BYRST02] (see also [BYGW99]) showed the impossibility of extracting randomness by making a single pass over the input. Their work generalizes to the lower bound that randomness extractors with single read/write stream and local memory of size s must make $\Omega(n/s)$ many passes.

Lower bounds are difficult to prove in the multi-stream model, since known single-stream lower bound techniques such as crossing sequence-like or communication complexity arguments cannot handle more than one stream. In this dissertation, we first introduced the lower bound technique ([PY14]) for big pseudorandom objects, e.g. pseudorandom generators, in the multi-stream model. However, these arguments rely on the fact that the input is stretched in the output, which rarely holds for extractors. Furthermore, unlike the deterministic pseudorandom generators, an extractor is a randomized algorithm whose computation depends on a uniform random seed. To that end, the lower bound proof for extractors requires more precise calculation combined

with averaging arguments. Another difference is that the lower bounds for extractors are not just super-constant $\Omega(1)$ many passes, but the unusual $\Omega(\log\log n)$. More surprisingly, such log-log lower bound turns out tight according to the matching construction of RRB. Note that lower bounds hold even for bit-fixing sources, and in fact for any constant number of bit-fixing sources; i.e. this lower bound applies to extractors that are simultaneously seeded and multiple-independent-source.

A technical problem in realizing a streaming extractor is that the multi-stream model is inherently limited in permuting input blocks. This is true even when the head can move in two directions over the streams [BH12, GHS09, GS05]. Our slightly-above-constant lower bound implies that at least a mild form of permutation is necessary. Indeed, by putting in perspective previous constructions (e.g. [RRV99, SU01, Tre99]) we observe that they all access the input bits in a permuted fashion, and this is true even after fixing the seed. However, these previous extractors make expensive operations when viewed as streaming algorithms. The same holds true for extractors from multiple independent sources, where access to the input also relies on accessing the input through streaming-wise-costly combinatorial objects such as error-correcting codes [DEOR04, Raz05]; e.g. multiplying a matrix $\mathbf{M}$ by a vector $\mathbf{x}$ which requires $\Omega(\log n)$ many passes or polynomial local memory by [GHS09].

There are practical efficiency issues in applying previous extractors on very big objects. Those extractors are not designed for very long input, and their running time are commonly polynomial in the input length n, e.g. $\mathrm{O}(n^2)$. In the context of extraction from Big Data, even quadratic running time becomes intolerable since n is too large. For instance, we observe that an implementation of Trevisan's extractor would spend more than $100,000$ years on a 20 GB input, which is estimated via polynomial fitting of experimental results on small input samples (Figure 3-2-B on page 87). In addition, the space used on the streams cannot grow quadratically either, which precludes the simulation of Log-space algorithms with $\mathrm{O}(\log n)$ passes over 2 streams [CY91, HS08]. As a comparison, RRB achieves streaming efficiency with running time $\mathrm{O}\left(n\log\log\frac{n}{\varepsilon}\right)$ and $\mathrm{O}(n)$ space on the streams.

Conceptually, RRB differs from known seeded extractors, e.g. *Trevisan's extrac-*

tor [Tre99] and its followups. The conceptual simplicity of RRB leads to efficiency in the multi-stream model.

In the beginning RRB applies "streaming friendly" permutations, which cost only $\mathrm{O}\left(\log\log\frac{n}{\varepsilon}\right)$ passes with an appropriately implementation. These mild permutations prepare the input for the last part of RRB, which is an one-pass local extractor that operates on every block with the same extractor and seed.

The last part of RRB alone, i.e. applying the same local extractor on every block, is proved to work correctly conditioned on guaranteed *next-block-min-entropy* [CG88, CMV13, SV86, Zuc96]. Here "guaranteed next-block-min-entropy" means that conditioned on the worst-case choice of all preceding blocks, there is still enough min-entropy left in each block (see Section 1.4 on page 14). Those previous works [CG88, CMV13, Zuc96] use the last part of RRB alone only under restricted conditions, e.g. extracting from multiple sources where each block is independently sampled. However, to process every block locally in $\mathsf{polylog}(n)$ memory, every sample must have size $\mathsf{polylog}(n)$ and hence there must be $\Omega\left(n/\mathsf{polylog}(n)\right)$ many independent samples. This is substantially different from the $\mathrm{O}\left(\log\frac{n}{\varepsilon}\right)$ many independent samples required by RRB for general weak sources (Theorem 3.17). At a retrospect, the major effort in RRB is to obtain next-block-min-entropy from as mild as possible input samples.

Finally, we note that RRB is an oblivious streaming extractor (i.e. after fixing the seed the head-move on the external streams depends only on n). Thus, RRB, which makes $\mathrm{O}\left(\log\log\frac{n}{\varepsilon}\right)$ passes, does not work for sources of min-entropy $k = \mathrm{o}(n)$, where $\Omega(\log n)$ many passes are needed (Theorem 4.11 on page 112).

Chapter 2

Cryptography in the Multi-Stream Model

In this chapter, we discuss cryptographic constructions in the multi-stream model. The exposition begins with necessary notations and formal context in Section 2.1, and a warm-up example for one-way functions in Section 2.3. The multi-stream constructions of pseudorandom generators and the PKE scheme are presented in Section 2.4 and Section 2.5. Finally, we summarize in Section 2.6. The lower bounds for cryptographic primitives are discussed in Chapter 4, Section 4.2.

2.1 Preliminary – Notation and Background

We use capital letters to denote random variables and sets, and in particular X, Y, Z for random variables and S, T for sets. For convenience we let $[n] \stackrel{\text{def}}{=} \{1, 2, 3, \ldots, n\}$. Calligraphic letters, e.g. $\mathcal{D}$, are used to denote probability distributions. We use $x \sim \mathcal{D}$ to denote that x is sampled from $\mathcal{D}$, and $x \in_R S$ when x is sampled uniformly from the set S.

We denote types of independent random variables by calligraphic letters such as $\mathcal{U}_n$ for the uniform distribution over $\{0,1\}^n$ and $\mathcal{U}_S$ for the uniform distribution over a set S. All distributions are over $\{0,1\}^n$ for some positive integer n unless otherwise specified. The *statistical distance* between two distributions X, Y over $\{0,1\}^n$ is

defined as $\mathcal{SD}(X,Y) \stackrel{\text{def}}{=} \frac{1}{2}\sum_{w\in\{0,1\}^n} \big|\Pr[X=w]-\Pr[Y=w]\big|$. We say that X is *ε-close to uniform* if $\mathcal{SD}\left(X,\mathcal{U}_{|X|}\right) \le \varepsilon$.

We use capital bold letters, e.g. $\mathbf{A}$, to denote matrices, and lower case bold letters, e.g. $\mathbf{x}$, for column vectors, and correspondingly $\mathbf{x}^T$ for row vectors. Let $\mathbb{Z}_q := \mathbb{Z}/q\mathbb{Z} = \{0,1,2,\cdots,q-1\}$ be the ring of integers with addition and multiplication modulo q.

Lowercase letters such as x, y are used for binary strings. By subscripting a string with a set, e.g. $x|_S$, we denote the subsequence of x restricted on the positions specified by S in increasing order, and the same holds for random variables. For example, $x|_{\{1,3,42\}} = (x_1, x_3, x_{42})$. Substrings of x are denoted by $\overline{x}_1, \overline{x}_2, \cdots$, and in particular $(\overline{x}_1, \ldots, \overline{x}_b)$ refers to a partition of x into b many consecutive parts, where each part $\overline{x}_i$ is called a *block of* x. Moreover, when x is sampled from a random variable X, for every $i \in [b]$ we denote by $\overline{X}_i$ the block of X where $\overline{x}_i$ is sampled from. To avoid ambiguity, brackets are used to index bits; e.g. $X_1[2]$ denotes the second bit of the random variable X_1.

Complexity theory related notation. In this dissertation all complexity classes are function classes (but we prefer to write e.g. NC^0 instead of FNC^0). Log-space denotes the set of all functions computable by a Turing machine (transducer) with a read-only input, $\mathrm{O}(\log n)$ large working tape, and a write-only output tape.

For every non-negative integer $i \in \mathbb{Z}^{\ge 0}$, NC^i denotes the set of functions computable by families of boolean circuits that for every input length n there is one poly-size circuit consists of constant fan-in gates and has $\mathrm{O}\left(\log^i n\right)$ depth. Since NC^0 circuits have constant depth, every output bit in an NC^0 circuit only depends on constant many input bits, regardless of the input length. Thus, NC^0 is also called "constant parallel time".

A family of circuits is (Log-space) uniform if there is a (Log-space) Turing machine such that on input 1^n for $n \in \mathbb{Z}^{\ge 0}$, it generates the description of the corresponding circuit in the family. In particular, every function in Log-space uniform NC^1 is also in Log-space [AB09].

In this chapter, we consider streaming algorithms for $(p,s,t) = (\mathrm{O}(1), \mathrm{O}(\log n), 2)$,

following Definition 1.1 on page 4, which means algorithms computable with Turing machines with $\mathrm{O}(\log n)$ local memory plus constant many sequential passes over 2 read/write streams.

Cryptographic primitives. We begin with *one-way function*, which is a very fundamental cryptographic primitive. Many other primitives are in particular one-way functions.

Definition 2.1. *$f : \{0,1\}^* \to \{0,1\}^*$ is a* (T, ε)-secure one-way function *for $T = T(n), \varepsilon = \varepsilon(n)$ if f is polynomial time computable and for all sufficiently large n, for every time T randomized algorithm $\mathcal{A}$, $\Pr_{y \sim f(\mathcal{U}_n)}[f\big(\mathcal{A}(y)\big) = y] < \varepsilon$.*

For simplicity, we omit (T, ε) for *computational security*, which refers to $T = n^{\omega(1)}, \varepsilon = n^{-\omega(1)}$. This also applies to the notation of pseudorandom generators below.

Definition 2.2. *A polynomial time computable function $G : \{0,1\}^* \to \{0,1\}^*$ is called a* (T, ε)-pseudorandom generator *if $\forall x, |G(x)| > |x|$ and $G(\mathcal{U}_n)$ is* (T, ε)-pseudorandom*, i.e. for sufficiently large n and for every randomized distinguisher $\mathcal{D}$ with running time T, $\left|\Pr[\mathcal{D}(G(\mathcal{U}_n)) = 1] - \Pr[\mathcal{D}(\mathcal{U}_{|G(1^n)|}) = 1]\right| < \varepsilon$.*

We also introduce the following public-key cryptographic primitive.

Definition 2.3. *A* public-key encryption (PKE) scheme *consists of three polynomial time algorithms for key-generation, encryption, and decryption respectively:*

i. **KeyGen** *takes the security parameter 1^n as input and it generates a public encryption key $\mathcal{PK}$ and a private decryption key $\mathcal{SK}$. Both of $\mathcal{PK}, \mathcal{SK}$ have length at least n, and n can be determined from $\mathcal{PK}, \mathcal{SK}$.*

ii. **Enc** *takes input $(\mathcal{PK}, m)$ and outputs a ciphertext $c = \mathbf{Enc}(\mathcal{PK}, m)$, for every m drawn from the message space M which may depend on $\mathcal{PK}$.*

iii. **Dec** *takes input $(\mathcal{SK}, c)$ and outputs m with overwhelming probability over the random choices of* **Enc**. *That is, $\Pr[\mathbf{Dec}(\mathcal{SK}, \mathbf{Enc}(\mathcal{PK}, m)) \neq m] \leq \mathsf{neg}(n)$.*

The key-generation **KeyGen** *and the encryption* **Enc** *are probabilistic polynomial time algorithms, while the decryption* **Dec** *is deterministic.*

We say that *a PKE system is streaming computable* if both **Enc** and **Dec** are $(\mathrm{O}(1), \mathrm{O}(\log n), \mathrm{O}(1))$-streaming algorithms. We do not require **KeyGen** to be a streaming algorithm since it only invokes at the beginning offline phase.

In a PKE system, *Indistinguishability under Chosen Plaintext Attack* (IND-CPA) security is defined through the following security experiment, which is a game between a *challenger* and an *adversary*:

i. The challenger runs **KeyGen** and uses its random choices to generate a public $\mathcal{PK}$ and a private $\mathcal{SK}$ key, and reveals the $\mathcal{PK}$ to the adversary.

ii. The adversary chooses two equal-length messages $\mathbf{x}_0$ and $\mathbf{x}_1$, and sends them to the challenger.

iii. The challenger flips an unbiased coin $b \in_R \{0, 1\}$, computes $\mathbf{c} = \mathbf{Enc}(\mathcal{PK}, \mathbf{x}_b)$ and gives the ciphertext $\mathbf{c}$ to the adversary.

iv. The adversary outputs $b' \in \{0, 1\}$ based on $\mathcal{PK}$ and $\mathbf{c}$, and it *wins* if and only if $b' = b$.

Here the adversary is an arbitrary probabilistic polynomial time algorithm. The PKE is *IND-CPA secure* if for every $c \in \mathbb{R}$, we have $\Pr[b' = b] < \frac{1}{2} + \frac{1}{N^c}$, when the message length $N = |\mathbf{x}_0| = |\mathbf{x}_1|$ is sufficiently large.

Note that there are no unconditional constructions for the above cryptographic primitives, since otherwise there must be $\mathsf{P} \neq \mathsf{NP}$. Therefore, constructions of these primitives must be based on intractability assumptions. In this dissertation, we use the general assumption that one-way functions exist in NC^1. This assumption covers many specific popular intractability assumptions, such as Integer Factorization (Section 1.2.2 on page 8), Learning With Errors (Assumption 2.18 on page 45) and Decoding Random Linear Codes (Assumption 2.21 on page 53).

Randomized encoding and cryptography in NC^0. The randomized encoding technique (defined below) was introduced in [AIK06a, AIK06b, AIK08, AIK09, AIK10] to show feasibilities of cryptography in NC^0.

Definition 2.4. *For a function* $f : \{0,1\}^n \to \{0,1\}^m$*, the function* $\hat{f} : \{0,1\}^n \times \{0,1\}^\rho \to \{0,1\}^{m'}$ *is a* randomized encoding *of* f *if the following conditions hold*

i. For every $x \in \{0,1\}^n$*, the output distribution* $\hat{f}(x,\mathcal{U}_\rho)$ *uniquely determines a* $f(x)$*, i.e.* $\hat{f}(x,r) \neq \hat{f}(x',r')$ *for any* r, r'*, as long as* $f(x) \neq f(x')$*.*

ii. The output distribution is fully determined by the encoded value $f(x)$*, i.e. if* $f(x) = f(x')$ *then* $\hat{f}(x,\mathcal{U}_\rho)$ *and* $\hat{f}(x',\mathcal{U}_\rho)$ *are identically.*

iii. $|\rho| = \mathsf{poly}(n)$ *and there are* $\mathsf{poly}(n)$*-time algorithms to decode* $f(x)$ *from any sample in* $\hat{f}(x,\mathcal{U}_\rho)$*, and to sample from* $\hat{f}(x,\mathcal{U}_\rho)$ *when given* $f(x)$*.*

This technique is used to transform a hard-to-compute function into a much easier one that is information-theoretically equivalent. Intuitively, (i) means that $\hat{f}(x,r)$ contains all information about $f(x)$, and (ii) asserts that $\hat{f}(x,\mathcal{U}_\rho)$ reveals no extra information about x other than the value of $f(x)$. Putting these two together we have that if f is a one-way function then $\hat{f}$ is also one-way [AIK06b].

	Streaming Model	NC^0
one-way function pseudorandom generator (PRG)	✓	✓
PKE (**Enc** & **Dec**)	✓	×
linear-stretch PRG	**?**	✓*
super-linear-stretch PRG	×	**?**

Table 2.1: **Cryptography in the Multi-Stream Model vs Cryptography in NC^0.** The multi-stream model refers to the $(\mathrm{O}(1), \mathrm{O}(\log n), \mathrm{O}(1))$-streaming algorithms in this table. Checkmark "✓" means there exists a construction, cross "×" means a lower bound, and question mark "**?**" means the problem remains open.
* The linear-stream pseudorandom generator in NC^0 is constructed from Alekhnovitch's assumption [Ale03].

Both cryptography in the multi-stream model and cryptography in NC^0 rely on randomized encodings, but they are incomparable in a number of places. There are obvious things streaming algorithms can do (e.g. compute the parity of n bits, or sample almost uniformly from **Sym**(5); see Definition 2.10 on page 34 and Section 2.3.1) but NC^0 cannot, while NC^0 functions can perform non-trivial permutations that streaming algorithms cannot. Furthermore, there are concrete technical separations between

the multi-stream model and highly parallel NC^0, as shown in Table 2.1. For example, there is IND-CPA secure PKE with streaming encryption and decryption, whereas no NC^0 decryption is possible[1].

2.1.1 Single-Stream One-Way Functions are Impossible

We prove the impossibility of one-way functions with in the data stream model in Theorem 2.5, which improves on [KGY89].

In [KGY89] it is shown that a streaming algorithm with local memory $\mathrm{O}(\log n)$, a write-only output tape, and a single (i) read-only stream over which it makes a constant number of passes cannot compute (ii) a pseudorandom generator of linear stretch. We improve by showing that the single stream can be (i) read-write and that we cannot compute (ii) a one-way function; i.e. in particular, a pseudorandom generator.

As usual we write $f : \{0,1\}^n \to \{0,1\}^m$ as a representative of a family of functions $f : \{0,1\}^* \to \{0,1\}^*$, which contains one function for each input length $n \in \mathbb{Z}^{>0}$.

Theorem 2.5. *Every function $f : \{0,1\}^n \to \{0,1\}^m$ that is computable by an $(\mathrm{O}(1), \mathrm{O}(\log n), 1)$-streaming algorithm (i.e. $\mathrm{O}(\log n)$ local memory and constant many passes over single stream) is invertible in polynomial time.*

Proof. Let $f : \{0,1\}^n \to \{0,1\}^*$ be computable by an algorithm $\mathcal{A}_f$ with $s \leq c \log n$ local memory and p passes over single external tape, where both c, p are constants. We construct a non-deterministic $\mathsf{Log\text{-}space}$ adversary $\mathcal{A}$ that inverts f. The intuition is that $\mathcal{A}$ first guesses $\mathcal{A}_f$'s local memory, then simulate the computation of p passes simultaneously while guessing the pre-image x, and finally check consistency of the simulation. We first present the algorithm $\mathcal{A}$ and then discuss its correctness and efficiency.

Let $x = (x_1, x_2, \cdots, x_n)$ be the input. Without loss of generality, we consider $\mathcal{A}_f$ makes exactly p passes from left to right and only invokes new passes after scanning the last non-blank cell. For simplicity, we prove for the case when $\mathcal{A}_f$ is not allowed

[1]The problem is still open for AC^0, where AC^0 is the unbounded-fan-in-gates analog of NC^0.

to write on any blank cells on the external tape, i.e. the stream size is restricted to n. The argument generalizes to larger stream size, since for single-stream algorithms the blow-up of stream size is at most $p \cdot 2^s = \mathsf{poly}(n)$ when p is constant and $s = \mathrm{O}(\log n)$. Furthermore, we assume that $\mathcal{A}_f$ maintains in its local memory an *output index* (e.g. yc in the following algorithm) which indicates the current output position.

Input: 1^n and y

Process:

1 Non-deterministically select $p+1$ local memory snapshots $C_1, \cdots, C_{p+1} \in \{0,1\}^s$, where C_1 is the initial memory and C_{p+1} is a legal terminal memory

2 Initialize $\mathsf{xc} \leftarrow 1, \mathsf{yc} \leftarrow 0$

3 $C'_i \leftarrow C_i$, for $i = 1, 2, \cdots, p$

4 **while** $\mathsf{xc} \leq n$ **do**

5 Guess x_{xc} and $z \leftarrow x_{\mathsf{xc}}$

6 **for** $j = 1$ **to** p **do**

7 Simulate $\mathcal{A}_f$ with C'_j on z, and update C'_j, z accordingly

8 $\mathsf{yc} \leftarrow$ the output index in C'_j

9 **if** $\mathcal{A}_f$ *outputs* $b \neq y_{\mathsf{yc}}$ **then** halt with "Failed"

10 **end**

11 $\mathsf{xc} \leftarrow \mathsf{xc} + 1$

12 **end**

13 **for** $i = 1$ **to** p **do**

14 **if** $C'_i \neq C_{i+1}$ **then** halt with "Failed"

15 **end**

16 Halt with "Succeed"

Algorithm 1: $\mathcal{A}$ – inverting f given input length n and output $y = f(x)$

Lines 1–3 are for initialization. Lines 4–12 are used to simulate the computation of $\mathcal{A}_f$ on guesses input $x_1, ..., x_n$, and line 8 compares the output of simulation to the target y in an online fashion. Lines 13–15 checks whether the simulation corresponds to a consistent execution of $\mathcal{A}$. If everything is consistent, line 16 reports success of

the non-deterministic simulation.

For correctness, we prove that when $\mathcal{A}$ succeeds, the non-deterministic guess of x must be a pre-image of y. Because for every $i = 1, 2, \ldots, p$, $\mathcal{A}$ checks in Line 14 whether C_{i+1} is exactly the memory after the computation of $\mathcal{A}_f$ with C_i and the guessed input x. As long as $\mathcal{A}$ passes this check, then Line 7 is guaranteed to simulate in parallel the computation of $\mathcal{A}_f$ in all p passes on the guessed input x. Consequentially in Line 9, $\mathcal{A}$ checks whether the output of the simulation is the same as y. Therefore, $\mathcal{A}$ succeeds if and only if the guessed input is a pre-image of y and $C_1, \cdots, C_{p+1}$ are corresponding local memory content before each single pass.

Regarding efficiency, the adversary $\mathcal{A}$ stores $2p + 1$ copies of $\mathcal{A}_f$'s local memory, i.e. $C_1, \cdots, C_{p+1}, C'_1, \cdots, C'_p$, together with one bit guessed input z, and constant many counters, i.e. the instruction counter, iteration counters i, j, and counters xc, yc for the input and output positions being processed. All these variables together consume at most $(2p + 1)s + 1 + \mathrm{O}(\log n) = \mathrm{O}(\log n)$ bits of local memory. Thus, $\mathcal{A}$ is computable by a non-deterministic $\mathsf{Log\text{-}space}$ Turing machine. By a standard technique, all configurations can be enumerated and organized with a $\mathsf{poly}(n)$-sized digraph. After that, there is a deterministic polynomial algorithm finding a path from the initial configuration to a success configuration (since $y = f(x)$, such a success path always exists). Moreover, the success path reveals the non-deterministic guesses of x, which is a desired pre-image.

In conclusion, f is invertible by a polynomial time algorithm and hence it is not a one-way function. □

The above adversary easily extends to the case where there is an extra write-only tape, since it checks output consistency in an online way (Line 9 in $\mathcal{A}$). Furthermore, it extends[2] to the case where the inverter is in uniform NC.

[2]We would like to thank Eric Allender for bringing this to our attention.

2.2 Warm up: Streaming Encoding of Universal Hash Functions

The universal family of hash functions is an important and widely used combinatorial object. For instance, it serves as an indispensable part in many generic constructions of pseudorandom generators from one-way functions (see [HILL99, HRV10, VZ12]). It is also applicable as a randomness extractor by Lemma 3.8 (page 63), though inefficient in seed length.

In this part, we give the formal definition of universal families of hash functions, and then present an implementation in the multi-stream model.

Definition 2.6. *Let S_n^m be a set of functions from $\{0,1\}^n$ to $\{0,1\}^m$ and H_n^m be a random variable uniformly distributed over S_n^m. We say that S_n^m is a* universal[3] family of hash functions *(or a* hashing family *for short) if it satisfies the following two conditions:*

i. Every function in S_n^m has a succinct bit-string representation (i.e. encoded as a string in $\{0,1\}^{\mathsf{poly}(n,m)}$) and can be efficiently evaluated, i.e. there is a polynomial time algorithm E which computes $E(h,x) = h(x)$ on input the binary representation of $h \in S_n^m$ and $x \in \{0,1\}^n$. This algorithm E is called the universal evaluator for S_n^m.

ii. For every $x, y \in \{0,1\}^n$ and $x \neq y$, the random variables $H_n^m(x)$ and $H_n^m(y)$ are independent and identically distributed to $\mathcal{U}_m$, i.e. for every $z_1, z_2 \in \{0,1\}^m$,

$$\Pr_{h \sim H_n^m}[h(x) = z_1 \wedge h(y) = z_2] = 1/2^{2m}$$

The definition of hash families does not carry any hardness requirement and there are explicit constructions. For example, a popular hashing family is the set of all affine transformations, i.e. $\left\{h(\mathbf{x}) = \mathbf{M}\mathbf{x} + \mathbf{b} \;\middle|\; \mathbf{M} \in \mathbb{Z}_2^{m \times n}, \mathbf{b} \in \mathbb{Z}_2^m\right\}$ where the algebra is over $\mathbb{Z}_2$. Every function in this family has a description of length $(m+1)n$. A more

[3]This is sometimes referred to as "2-universal" since condition ii talks about independence between two elements.

succinct hashing family is obtained with affine transformations defined by *Toeplitz matrices*, which are matrices that have invariant values on each of its diagonals. In the Toeplitz hashing family, every function is described with $2m + n - 1$ bits.

Now, we present a universal family of linear hash functions in the multi-stream model. More specifically, we construct a perfect randomized encoding scheme in [PY14] for linear hash functions[4], i.e. $h(\mathbf{x}) = \mathbf{Mx}$ for $\mathbf{M} \in \mathbb{Z}_2^{n\times n}$. This encoding scheme achieves streaming efficiency while evaluating information-theoretically the same function as before being encoded. Note that although the argument is for random linear functions, everything is naturally generalized to the family of Toeplitz functions.

Claim 2.7. *Let S_n^m be a family of linear hash functions, and for every $h \in S_n^m$, $h : \{0,1\}^n \to \{0,1\}^m$ has the form $h(\mathbf{y}) = \mathbf{y}^T\mathbf{H}$, where $\mathbf{H} \in \{0,1\}^{n\times m}$ is the matrix specified by h. Then, every $h \in S_n^m$ has a randomized encoding $\hat{h} : \{0,1\}^n \times \{0,1\}^{(n-1)m} \to \{0,1\}^{nm}$, such that $S^{nm}_{(n-1)m+n} = \widehat{S_n^m} = \left\{\hat{h} | h \in S_n^m\right\}$ defines a universal family of streaming computable hash functions from $(n-1)m+n$ bits to nm bis. That is, there is a streaming computable universal evaluator $\hat{E} : \widehat{S_n^m} \times \{0,1\}^n \times \{0,1\}^{(n-1)m} \to \{0,1\}^{nm}$ for $\widehat{S_n^m}$ such that $\hat{E}(\hat{h}, \mathbf{y}, \mathbf{r}) = \hat{h}(\mathbf{y}, \mathbf{r})$. In particular, we say that $\hat{h}$ is a* streaming randomized encoding *of h.*

Proof. Suppose $\mathbf{y} = (y_1, \cdots, y_n)$, and $\mathbf{H}$ be as follows:

$$\mathbf{H} = \begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,m} \\ \vdots & \cdots & \cdots & \vdots \\ h_{n,1} & h_{n,2} & \cdots & h_{n,m} \end{bmatrix} \tag{2.1}$$

As a result, we have $\mathbf{y}^T\mathbf{H} = (\sum_{i=1}^n y_i h_{i,1}, \sum_{i=1}^n y_i h_{i,2}, \cdots, \sum_{i=1}^n y_i h_{i,m})$, where every output bit depends on all n bits in y. The summation over $\mathbb{Z}_2$ is equivalent to parity operation. In order to reduce the locality of the product $\mathbf{y}^T\mathbf{H}$, we introduce a sequence of random bits $r \in \{0,1\}^{n\times m}$ to partition every term $\sum_{i=1}^n y_i h_{i,j}$ into smaller "masked

[4]For cryptographic applications it suffices to consider linear functions rather than affine functions, though the output, without adding a random vector $\mathbf{b}$, is not uniformly distributed when $\mathbf{x} = \mathbf{0}$. Note that all the arguments for linear functions generalize to affine functions.

pieces", for each $j = 1, 2, \cdots, m$.

Thus, we design $\hat{h} : \{0,1\}^n \times \{0,1\}^{(n-1)\times m} \to \{0,1\}^{n\times m}$ as

$$\begin{aligned}\hat{h}(\mathbf{y}, \mathbf{r}) = \langle & y_1 h_{1,1} + r_{1,1}, && \cdots, && y_1 h_{1,m} + r_{1,m}, \\ & y_2 h_{2,1} + r_{1,1} + r_{2,1}, && \cdots, && y_2 h_{2,m} + r_{1,m} + r_{2,m}, \\ & \vdots && \cdots \\ & y_n h_{n,1} + r_{n-1,1}, && \cdots, && y_n h_{n,m} + r_{n-1,m} \rangle \end{aligned} \tag{2.2}$$

It is easy to verify that $\hat{h}$ is a perfect randomized encoding of the function $h(\mathbf{y}) = \mathbf{y}^T\mathbf{H}$, and $\hat{h}(\mathbf{y}, \mathbf{r})$ uniformly distributes over all possible encodings of $h(\mathbf{y})$.

We complete the proof with the realization of a universal evaluator $\hat{E}$ for $\hat{h}$ in an $(\mathrm{O}(1), \mathrm{O}(\log n), 2)$-streaming algorithm. By scanning $\mathbf{y}$ and $\mathbf{H}$ in a row-first order, $\hat{E}$ fills in all terms of the form $y_i h_{i,j}$. Then, $\hat{E}$ completes the computation of $\hat{h}(\mathbf{y}, \mathbf{r})$ (as represented in (2.2)) by making another two passes over $\mathbf{r}$ while scanning $y_i h_{i,j}$'s simultaneously. □

2.3 Streaming One-way Functions

We present a generic compiler (Section 2.3.2) that maps every $f \in \mathsf{NC}^1$ to its randomized encoding $\hat{f}$ in the multi-stream model. Due to a very useful coincidence regarding the specific encoding we use, we get $\hat{f}$ computable with 2 streams (the reader is encouraged to think ahead to see where the issue is). Corollary 2.9 immediately follows Theorem 2.8 by [AIK06b].

Theorem 2.8. *Every function $f \in \mathsf{NC}^1$ has a randomized encoding function $\hat{f}$ which is oblivious streaming computable with 5 passes.*

Corollary 2.9. *A streaming one-way function exists if there is a one-way function in Log-space (defined on page 24).*

Here is an advanced remark. The construction in the proof of Theorem 2.8 relies on a specific randomized encoding that also causes a polynomial blow-up compared

with the regular output length in Barrington's Theorem (page 34).

2.3.1 Background: NC^1 to width-5 Branching Programs

Now, we introduce the definition of a bounded-width permutation branching program. This model is interesting for streaming computation because (i) it is completely sequential, (ii) every step depends only a single input bit, and (iii) it is powerful enough to simulate functions in other classes (cf. [Bar86]).

Definition 2.10. *A* width-w permutation branching program *consists of a sequence of* $m = m(n)$ instructions $B_n = (s_1, \langle j_1, \sigma_1, \tau_1\rangle) \cdots (s_m, \langle j_m, \sigma_m, \tau_m\rangle)$, *for* $j_i \in [n] = \{1, 2, \cdots, n\}$, $\sigma_i, \tau_i \in \mathbf{Sym}(w)$. *Here* $\mathbf{Sym}(w)$ *is the group of permutations over* $[w] = \{1, 2, \cdots, w\}$. *On input* $x = (x_1, \cdots, x_n) \in \{0,1\}^n$, B_n *is evaluated as* $B_n(x) = s_1 \cdot s_2 \cdot \cdots \cdot s_m$, *where* $s_i = \sigma_i$ *if* $x_{j_i} = 1$ *and* $s_i = \tau_i$ *if* $x_{j_i} = 0$.

A function $f : \{0,1\}^n \to \{0,1\}$ is recognized by B_n if there exists a cycle $\theta \in \mathbf{Sym}(w)$, such that $\forall x \in \{0,1\}^n$, $B_n(x) = \theta$ when $f(x) = 1$, and $B_n(x) = e$ is the identity permutation when $f(x) = 0$.

Permutation branching programs can compute many interesting functions. In particular, width-5 branching program is sufficient for every function in NC^1, as shown in the following theorem. This theorem holds as well for the Log-space uniform case, i.e. all the instructions are generated by a Log-space algorithm (transducer) on input of 1^n.

Theorem 2.11 (Barrington's Theorem [Bar86])**.** *Any boolean function* f *computable by a family of depth* d *and fan-in 2 circuits can be recognized by a family of width-*5 *permutation branching programs with* $m \leq 4^d$. *In particular,* $m = \mathsf{poly}(n)$ *for* $f \in \mathsf{NC}^1$ *and input length* n.

Thus, evaluating $f : \{0,1\}^n \to \{0,1\}$ on input x reduces to deciding whether $B_n(x) = s_1 s_2 \cdots s_m$ is the identity. Let $\hat{f} : \{0,1\}^n \times \mathbf{Sym}(5)^{m-1} \to \mathbf{Sym}(5)^m$ be defined as

$$\hat{f}(x; r) = \langle \pi_1, \cdots, \pi_m \rangle = \langle s_1 r_1, r_1^{-1} s_2 r_2, \cdots, r_{m-2}^{-1} s_{m-1} r_{m-1}, r_{m-1}^{-1} s_m \rangle$$

where $r_i \in_R \mathbf{Sym}(5)$, s_i follows the i-th instruction in B_n and m is the length of B_n. Then, $\hat{f}$ is a randomized encoding of f, since $\langle \pi_1, \cdots, \pi_m \rangle$ uniformly distributes over $\mathbf{Sym}(5)^m$ conditioned on $\pi_1 \pi_2 \cdots \pi_m = s_1 s_2 \cdots s_m$.

We define $\mathbf{Sample} : \{0,1\}^q \to \mathbf{Sym}(5)$ to be the algorithm that samples $r_i \in_R \mathbf{Sym}(5)$ within statistical distance $2^{-\Omega(q)}$ using $q = q(n)$ (read-once) random bits. Then, every permutation in $\mathbf{Sym}(5)$ is identified by its unique binary ID in $\{0,1\}^7$. Thus, $\hat{f}$ is represented in binary as $\widehat{f} : \{0,1\}^n \times (\{0,1\}^q)^{m-1} \to (\{0,1\}^7)^m$ that induces a loss of at most $2^{-\Omega(q(n))}$ in the output distribution. It only remains to make $\widehat{f}$ streaming computable. The issue is that non-consecutive s_i's may be arbitrarily associated with the same input bit, so we must do something about this.

2.3.2 Streaming Computable Randomized Encoding

Our streaming algorithm for $\hat{f}$ is based on the following observations:

i. fixing any poly-time invertible permutation ψ over $\{1, \ldots, m\}$, then for $\hat{f}(x; r) = \langle \pi_1, \cdots, \pi_m \rangle$, the permuted function $g(x; r) = \langle \pi_{\psi(1)}, \cdots, \pi_{\psi(m)} \rangle$ is a one-way function as long as $\hat{f}$ is a one-way function;

ii. a permutation branching program (e.g. B_n) recognizes exactly the same function after inserting *dummy instructions* in the form of $(s, \langle j, e, e \rangle)$; in other words, the program $(s_1, \langle j_1, \sigma_1, \tau_1 \rangle) \cdots (s_m, \langle j_m, \sigma_m, \tau_m \rangle)$ recognizes essentially the same function as $(s_1, \langle j_1, \sigma_1, \tau_1 \rangle) \cdots (s, \langle j, e, e \rangle) \cdots (s_m, \langle j_m, \sigma_m, \tau_m \rangle)$.

By the second observation, for the length m branching program B_n that recognizes f, we may replace first instruction of B_n, i.e. $\big(s_1, \langle j_1, \sigma_1, \tau_1 \rangle\big)$, with one real instruction as before together with $n-1$ dummy instructions (as place holders), i.e. $\big(s'_1, \langle 1, e, e \rangle\big) \cdots \big(s'_{j_1}, \langle j_1, \sigma_1, \tau_1 \rangle\big) \cdots \big(s'_n, \langle n, e, e \rangle\big)$, where $s'_{j_1} = s_1$ and $s'_i = e$ for $i \neq j_1$. Thus, these n instructions are functionally equivalent to the original instruction $\big(s_1, \langle j_1, \sigma_1, \tau_1 \rangle\big)$, since $s'_1 \cdots s'_n = s_1$ on every input. This kind of inefficient replacement bring us the advantage that $\forall i \in \{1, 2, \ldots, n\}$, the permutation s'_i depends on x_i with exactly the same index.

We apply the similar idea for the rest $m-1$ instructions, and introduce $m(n-1)$

dummy permutations after all. Then, we get a length $mn = \mathsf{poly}(n)$ new branching program B'_n whose $s'_1, \cdots, s'_{mn}$ obliviously depend on input bits, i.e. the j-th permutation s'_j depends on x_i for $i = j \pmod n$. In what follows, we abuse notation and use B_n and s_i instead of B'_n and s'_i for simplicity.

The above treatment is both conceptually and technically (modulo our randomized encoding) unavoidable. Conceptually, without obliviousness it would have been complicated to aggregate all permutations depending on the same input bit at the right position. Technically, the issue of making the branching program oblivious and consequently blowing up its length is not an aside issue. For example, if this could have been avoided then it would have been the case that applying our techniques we can obtain streaming linear stretch pseudorandom generators, given that a linear stretch pseudorandom generator exists in NC^0. Note that we have left as an open question the existence of such streaming pseudorandom generators. In particular, we do not know how to achieve this even when starting from Alekhnovitch's assumption[5] [Ale03, AIK08], which is sufficient for constructing linear stretch pseudorandom generators in NC^0. All these signify that removing the blow-up induced by the transformation to an oblivious one is a conceptually difficult task with non-trivial implications.

Now, after introducing all the necessary dummy instructions, we compute the s_i's in the following order with a single pass: $s_1, s_{n+1}, \cdots, s_{mn-n+1}, s_2, s_{n+2}, \cdots, s_{mn-n+2}, \cdots\cdots, s_n, s_{2n}, \cdots, s_{mn}$ (sorted by their dependency on x, which coincide the subscripts modular n). Then, for $f : \{0,1\}^n \to \{0,1\}$, we apply the first observation to construct the oblivious randomized encoding $\widehat{f} : \{0,1\}^n \times (\{0,1\}^q)^{mn-1} \to (\{0,1\}^7)^{mn}$ as follows

$$\widehat{f}(x; y_1, \ldots, y_{mn-1}) = \left\langle s_1 r_1,\ r_n^{-1} s_{n+1} r_{n+1}, \cdots, r_{mn-n}^{-1} s_{mn-n+1} r_{mn-n+1} \cdots, r_{mn-1}^{-1} s_{mn} \right\rangle$$

[5]For every $m(n) = \mathrm{O}(n)$ and $\mu \in (0,1)$, there exits a positive integer ℓ and a family of binary matrices $\{\mathbf{M}_n\}_{n \in \mathbb{Z}^+}$, where $\mathbf{M}_n$ is an $m(n) \times n$ matrix with exactly ℓ ones in each row, such that the distribution of $\mathbf{M}_n \times \mathcal{U}_n + \mathbf{e}_\mu$ is computational indistinguishable from $\mathbf{M}_n \times \mathcal{U}_n + \mathbf{e}_{\mu+1/m(n)}$, where $\mathbf{e}_\mu$ is a random error vector whose entry equals one with probability μ (independently from other entries) and $\mathbf{e}_{\mu+1/m(n)}$ is defined similarly.

where $r_i = \mathbf{Sample}(y_i)$, r_i^{-1} is the inverse of r_i, and s_i is a function of $x_{(i \mod n)}$ for $i = 1, 2, \cdots, mn$.

When $f(x) = \langle f_1(x), f_2(x), \cdots, f_{\ell(n)}(x)\rangle$ has $\ell(n)$ output bits, we design $\hat{f} = \langle \hat{f}_1, \cdots, \hat{f}_{\ell(n)}\rangle$, which consists of an individual randomized encoding $\hat{f}_i$ one for each f_i. It is not too hard to globally rearrange the output bits and obtain the final streaming computable function $\widehat{f}$.

Proof of Theorem 2.8. Consider the function $f(x) = \langle f_1(x), f_2(x), \cdots, f_{\ell(n)}(x)\rangle$ where $f_i : \{0,1\}^n \to \{0,1\}$ is in NC^1 for every $i \in \{1, 2, \cdots, \ell(n)\}$. In an analog manner as in the construction in Section 2.3.2, let $s_1^{(i)}, s_2^{(i)}, \cdots, s_{mn}^{(i)}$ follow the permutation branching program of f_i by Barrington's theorem, and let $r_1^{(i)}, r_2^{(i)}, \cdots r_{mn-1}^{(i)}$ be sampled from $\mathbf{Sym}(5)$ with the random input r. Thus, every f_i has a randomized encoding

$$\hat{f}_i(x, r) = \langle s_1^{(i)} r_1^{(i)}, (r_n^{(i)})^{-1} s_{n+1}^{(i)} r_{n+1}^{(i)}, \cdots\cdots, (r_{mn-n-1}^{(i)})^{-1} s_{mn-n}^{(i)} r_{mn-n}^{(i)}, (r_{mn-1}^{(i)})^{-1} s_{mn}^{(i)}\rangle.$$

Now, we turn to the construction of $\hat{f}$. Putting $\hat{f}_i$'s together and rearrange terms, we define

$$\begin{aligned}
\hat{f}(x, r) = \big\langle\ & s_1^{(1)} r_1^{(1)}, && \cdots, && s_1^{(\ell(n))} r_1^{(\ell(n))}, \\
& (r_n^{(1)})^{-1} s_{n+1}^{(1)} r_{n+1}^{(1)}, && \cdots, && (r_n^{(\ell(n))})^{-1} s_{n+1}^{(\ell(n))} r_{n+1}^{(\ell(n))}, \\
& \vdots && \cdots, && \vdots \\
& (r_{mn-n}^{(1)})^{-1} s_{mn-n+1}^{(1)} r_{mn-n+1}^{(1)}, && \cdots, && (r_{mn-n}^{(\ell(n))})^{-1} s_{mn-n+1}^{(\ell(n))} r_{mn-n+1}^{(\ell(n))}, \\
& \vdots && \cdots, && \vdots \\
& (r_{mn-n-1}^{(1)})^{-1} s_{mn-n}^{(1)} r_{mn-n}^{(1)}, && \cdots, && (r_{mn-n-1}^{(\ell(n))})^{-1} s_{mn-n}^{(\ell(n))} r_{mn-n}^{(\ell(n))}, \\
& (r_{mn-1}^{(1)})^{-1} s_{mn}^{(1)}, && \cdots, && (r_{mn-1}^{(\ell(n))})^{-1} s_{mn-n}^{(\ell(n))}\ \big\rangle
\end{aligned} \tag{2.3}$$

The i-th column in $\hat{f}(x, r)$ consists of exactly the same elements as those in the output of $\hat{f}_i(x, r)$. So, $\hat{f}$ is just a permutation of $\langle \hat{f}_1, \cdots, \hat{f}_{\ell(n)}\rangle$, and hence forms a randomized encoding of $f = \langle f_1, \cdots, f_{\ell(n)}\rangle$. Clearly, there exists a polynomial time algorithm sampling an identical distribution to $\hat{f}(x, r)$ when given $f(x)$. So that $\hat{f}$

inherits one-wayness from f.

We provide a streaming algorithm for $\hat{f}$. Suppose at the beginning the input is written in one external stream while the other is blank. For efficiency we assume the input is written as follows

$$\langle x_1, r_1^{(1)}, r_1^{(2)}, \cdots, r_{mn-n+1}^{\ell(n)}; x_2, r_2^{(1)}, \cdots, r_{mn-n+2}^{\ell(n)}; \cdots; x_n, r_n^{(1)}, \cdots, r_{mn-n}^{\ell(n)} \rangle$$

By scanning both streams once, we can compute the following results:

$$\langle s_1^{(1)} r_1^{(1)}, \cdots, s_{mn-n+1}^{\ell(n)} r_{mn-n+1}^{\ell(n)}; \cdots\cdots; s_n^{(1)} r_n^{(1)}, \cdots, s_{mn-n}^{\ell(n)} r_{mn-n}^{\ell(n)} \rangle$$

$$\langle (r_1^{(1)})^{-1}, \cdots, (r_{mn-n+1}^{\ell(n)})^{-1}; (r_2^{(1)})^{-1}, \cdots, (r_{mn-n+2}^{\ell(n)})^{-1}; \cdots; (r_n^{(1)})^{-1}, \cdots, (r_{mn-n}^{\ell(n)})^{-1} \rangle$$

The content of the first tape is computable because the permutation branching programs are log-space uniform and in every block the permutations only depend on one input bit, e.g. $s_1^{(1)}, \cdots, s_{mn-n+1}^{\ell(n)}$ only depend on x_1. The second tape contains simply the inverse permutations of those written on the input tape. Note that in (2.3) the inverse permutations appear nearly in the same order as what we have now on the second tape, i.e. it requires moving the $(r_n^{(1)})^{-1}, \cdots, (r_{mn-n}^{\ell(n)})^{-1}$ part to the beginning. Therefore, by scanning the second tape twice and together the first stream once, we can computed exactly $\hat{f}(x, r)$ as in (2.3).

Thus, we generalize the construction to the multi-bit case and get a one-way function computable with totally 5 passes over 2 external streams in the streaming model.

One may suggest the input given in a different order. However it does not really cause problems, since we do not care the order of random permutations r and it is also reasonable to rename the bits in x with respect to their order in the input. Moreover, with constant many passes we can always transform the input into the form as assumed. □

2.4 Streaming Pseudorandom Generators

The encoding in Section 2.3.2 does not preserve pseudorandomness, simply because the encoding is over $\mathbf{Sym}(5)$ but $|\mathbf{Sym}(5)| = 120$ is not a power of 2. In fact, Barrington's theorem (Theorem 2.11, p. 34) holds for every non-solvable $\mathbf{Sym}(w)$, as long as $w \geq 5$. However, there is no $k \in \mathbb{Z}$ such that $2^k | w!$, and hence it turns out quite non-trivial to convert the output encoding in $\mathbf{Sym}(w)$ into pseudorandom binary bits. Yet, we provide a rather technical adaptation of [VZ12] to build streaming pseudorandom generators directly from any streaming one-way function f.

Theorem 2.12. *Let $f : \{0,1\}^n \to \{0,1\}^m$ be a streaming one-way function. Then, there is a streaming computable pseudorandom generator G requiring 2 additional passes to the streaming algorithm for $\langle f^{(1)}, \cdots, f^{(\ell t)} \rangle$, for ℓ, t defined as below.*

Moreover, if $m = \mathrm{O}(n)$, then $\ell = n/\log n, t = \mathrm{O}\left(n^2 \log^2 n\right)$, and the seed length of G is $\mathrm{O}\left(n^6 \log^3 n\right)$.

Corollary 2.9 and Theorem 2.12 yield the following corollary that shows the feasibility of pseudorandom generators in the multi-stream model.

Corollary 2.13. *If there is a one-way function in NC^1 or Log-space, then there exists a pseudorandom generator which is streaming computable with 7 passes.*

In fact, the construction in Section 2.3.2 gives an oblivious streaming one-way function, which implies that evaluating polynomial many copies of f does not need more passes than computing a single copy of f. The construction in [VZ12, HRV10] is necessary as opposed to [HILL99], since it bypasses the dependency on certain parameters, i.e. the entropy of $f(\mathcal{U}_n)$, that we cannot handle with streaming algorithms. In particular, we cannot enumerates all possible values of $\mathsf{H}[f(\mathcal{U}_n)]$ (even up to some fixed accuracy) as in [HILL99], when the entropy is not efficiently computable.

The argument in [VZ12, HRV10] relies on the following computational analog of next-block-entropy and next-block-min-entropy (page 14).

Definition 2.14. *Let random variable $X = (X_1, \ldots, X_m)$ be the concatenation of m random variables, where X and m depend on a security parameter n. For every $T =$*

$T(n)$, $\varepsilon = \varepsilon(n)$, *we say that every block of* X *has* (T, ε) next-block-pseudo-entropy *(resp.* next-block-pseudo-min-entropy*)* k *if there exists a set of random variables* $Y = (Y_1, \ldots, Y_m)$ *such that for every* $i \in [m]$,

- $\mathsf{H}[Y_i | X_1, \ldots, X_{i-1}] \geq k$ *(resp.* $\mathsf{H}_\infty[Y_i | X_1, \ldots, X_{i-1}] \geq k$*);*
- Y_i *is* (T, ε)*-indistinguishable from* X_i *conditioned on* $X_1, \ldots, X_{i-1}$*, i.e. for every*[6] *distinguisher* $\mathcal{D}$ *that is a randomized algorithm of running time* T*,*

$$\left| \Pr\left[\mathcal{D}\left(X_1, \ldots, X_i\right) = 1\right] - \Pr\left[\mathcal{D}\left(X_1, \ldots, X_{i-1}, Y_i\right) = 1\right] \right| \leq \varepsilon$$

Proof of Theorem 2.12. In this proof, we first briefly introduce the main steps of the pseudorandom generator construction in [VZ12] (also [HRV10]), and then present the modification and implementation in the multi-stream model.

There are four steps in the construction:

i. **Next-Block-Pseudo-Entropy Generator:** Given any one-way function f, we construct a next-block-pseudo-entropy generator $G_{nb} : \{0,1\}^n \to \{0,1\}^m$ that transfers n-bit input to an output with next-block-pseudo-entropy $k = n + \Delta$, where $\Delta \geq \log n$.

ii. **Entropy Equalizer:** We evaluate G_{nb} on ℓ independent seeds and concatenate their output, and then we shift it by $j \in_R [m]$ blocks with $EQ : [m] \times \{0,1\}^{\ell m} \to \{0,1\}^{(\ell-1)m}$. Thus, every block has the same amount of next-block-pseudo-entropy $\alpha = k/m$.

iii. **Converting Shannon Entropy to Min-Entropy and Amplifying the Gap:** We take t-fold repetition of EQ, which concatenate their outputs within each block, to convert Shannon entropy to min-entropy (Definition 1.2 on page 13).

iv. **Randomness Extraction:** Finally, we pick a single random universal hash function to extract pseudo-min-entropy from each block, and concatenate the results as the final output.

[6] This definition is for non-uniform distinguishers. In the uniform setting distinguishers must be given an oracle that produces random samples of X, Y. See [VZ12] for more detailed discussion.

For step (i), [HRV10] use the construction $G_{nb}(\mathbf{s}, h) = \langle f(\mathbf{s}), h, h(\mathbf{s})_1, \cdots, h(\mathbf{s})_n \rangle$, which requires a randomized encoding of the hash function h when compiling to streaming algorithms and hence results in additional passes and less seed efficiency. The follow-up work [VZ12] suggests that indeed $G_{nb}(\mathbf{s}) = \langle f(\mathbf{s}), \mathbf{s} \rangle$ has next-block-pseudo-entropy $n + \log n$, when $\mathbf{s} \sim \mathcal{U}_n$. Such characterization significantly simplifies the construction as well as proof of the streaming pseudorandom generator.

We first introduce the following lemma for step (i).

Lemma 2.15 (Theorem 1.5 in [VZ12])**.** *If $f : \{0,1\}^n \to \{0,1\}^{m'}$ is a one-way function, then for $\mathbf{s} \sim \mathcal{U}_n$, $G_{nb}(\mathbf{s}) = \langle f(\mathbf{s}), \mathbf{s} \rangle$ has next-block-pseudo-entropy $n + \log n$.*

Such G_{nb} is trivially streaming computable as long as f is a streaming one-way function. Letting $m' = m - n$, we get the generator $G_{nb} : \{0,1\}^n \to \{0,1\}^m$ with (T', ε')-next-block-pseudo-entropy $k = n + \log n$, where $T', 1/\varepsilon'$ are both super-polynomial in n.

Secondly, let $EQ : \{0,1\}^{d_\ell} \to \{0,1\}^{m_\ell}$ follow the construction in [HRV10] with $\ell = n/\log n, d_\ell = \log m + m\ell, m_\ell = (\ell - 1)m$, such that for $j \in [m]$ and $z^{(1)}, \cdots, z^{(\ell)} \in \{0,1\}^m$,

$$EQ\left(j, z^{(1)}, \cdots, z^{(\ell)}\right) := \left\langle z_j^{(1)}, \cdots, z_m^{(1)}, \cdots, z_1^{(\ell)}, \cdots, z_{j-1}^{(\ell)} \right\rangle$$

For $z^{(1)}, \cdots, z^{(\ell)}$ independently drawn from $G_{nb}(\mathcal{U}_n)$ and $j \sim J = \mathcal{U}_{[m]}$, we denote the output distribution of EQ by $\mathcal{X}$,

$$\mathcal{X} = EQ\left(J, G_{nb}(\mathcal{U}_n)^{(1)}, \cdots, G_{nb}(\mathcal{U}_n)^{(\ell)}\right)$$

where every single-bit block of $\mathcal{X}$ has $(T' - \mathrm{O}(\ell m), \ell\varepsilon')$-next-block-pseudo-entropy at least $\alpha = k/m$ by Claim 5.2 in [HRV10].

EQ is streaming computable with the same number of passes as evaluating G_{nb} on ℓ independent seeds.

Then, we consider the t-fold of $\mathcal{X} \in \{0,1\}^{m_\ell}$ for the third step as follows, where

$t = \mathrm{O}\left(n^2 \log^2 n\right)$.

$$\mathcal{Y} = \mathcal{X}^t = \left\langle \left(\mathcal{X}_1^{(1)}, \cdots, \mathcal{X}_1^{(t)}\right), \cdots, \left(\mathcal{X}_{m_\ell}^{(1)}, \cdots, \mathcal{X}_{m_\ell}^{(t)}\right) \right\rangle$$

By Claim 5.3 in [HRV10], for every $i \in [m_\ell]$, the t-bit block $\left(\mathcal{X}_i^{(1)}, \cdots, \mathcal{X}_i^{(t)}\right)$ in $\mathcal{Y}$ has $(T' - \mathrm{O}(m_\ell t), t^2(\ell\varepsilon' + 2^{-\kappa} + 2^{-ct}))$-next-block-pseudo-*min*-entropy α_t, for a universal constant c, every $\kappa = \kappa(n) > 0$, and $\alpha_t = \alpha_t(n) = t \cdot \alpha - \mathrm{O}\left(\log t \cdot \sqrt{t \cdot \kappa}\right)$. In particular, we set $\kappa = \log^2 n$.

The random variable $\mathcal{Y}$ is streaming computable when represented in the form $\langle \mathcal{X}^{(1)}, \cdots, \mathcal{X}^{(t)} \rangle$.

Finally, we want to evaluate the *same* hash function (randomly drawn from $S_t^{\alpha_t - \kappa}$) on every block of $\mathcal{Y}$, to extract $\alpha_t - \kappa$ almost uniform bits from the α_t bits next-block-min-entropy. [HRV10, VZ12] relies on Ext defined as

$$\mathsf{Ext}\left(\mathcal{Y}, H_t^{\alpha_t - \kappa}\right) = \left\langle H_t^{\alpha_t - \kappa}, H_t^{\alpha_t - \kappa}(\mathcal{Y}_1), \cdots, H_t^{\alpha_t - \kappa}(\mathcal{Y}_{m_\ell}) \right\rangle$$

Although Ext is not streaming computable due to the hash functions, we can fix this problem by first relaxing one hash function for all blocks to a random hash function for each block, and then using the randomized encoding hash family $\widehat{S_t^{\alpha_t - \kappa}}$ as in (2.2) instead. We define the new function $\widehat{\mathsf{Ext}}$ as follows.

$$\widehat{\mathsf{Ext}}\left(\mathcal{Y}, \mathbf{H}; \mathbf{R}\right) = \left\langle \mathbf{H}, \widehat{\mathbf{H}^{(1)}}(\mathcal{Y}_1, \mathbf{R}^{(1)}), \cdots, \widehat{\mathbf{H}^{(m_\ell)}}(\mathcal{Y}_{m_\ell}, \mathbf{R}^{(m_\ell)}) \right\rangle$$

where $\mathbf{H} = \left(\mathbf{H}^{(1)}, \cdots, \mathbf{H}^{(m_\ell)}\right) \in_R \{0,1\}^{m_\ell \cdot t(\alpha_t - \kappa)}$ consists of the description of m_ℓ random linear hash function, and $\mathbf{R} = \left(\mathbf{R}^{(1)}, \cdots, \mathbf{R}^{(m_\ell)}\right) \in_R \{0,1\}^{m_\ell \cdot (t-1)(\alpha_t - \kappa)}$ are the random bits used in the encoding.

$\widehat{\mathsf{Ext}}$ is streaming computable with only 2 passes after properly reordering its output. When scanning $\mathcal{X}^{(i)}$ on one stream and $\mathbf{H}_i^{(1)}, \mathbf{R}_{i-1}^{(1)} + \mathbf{R}_i^{(1)}, \cdots, \mathbf{H}_i^{(m_\ell)}, \mathbf{R}_{i-1}^{(m_\ell)} + \mathbf{R}_i^{(m_\ell)}$ on the other, compute $\mathcal{X}_1^{(i)}\mathbf{H}_i^{(1)} + \mathbf{R}_{i-1}^{(1)} + \mathbf{R}_i^{(1)}, \cdots, \mathcal{X}_{m_\ell}^{(i)}\mathbf{H}_i^{(m_\ell)} + \mathbf{R}_{i-1}^{(m_\ell)} + \mathbf{R}_i^{(m_\ell)}$, where $\mathbf{H}_i^{(j)}$ is the i-th row of $\mathbf{H}^{(j)}$ and respectively $\mathbf{R}_i^{(j)}$ is the i-th row of $\mathbf{R}^{(j)}$ for $j \in [m_\ell]$. The sums of $\mathbf{R}_{i-1}^{(m_\ell)} + \mathbf{R}_i^{(m_\ell)}$ can be prepared during the passes used to compute G_{nb}.

To incorporate $\mathbf{H}^{(1)}, \cdots, \mathbf{H}^{(m_\ell)}$ in a single function $\mathbf{H}$, we define $\widetilde{\mathcal{Y}}$ for $\mathcal{Y} = \langle \mathcal{Y}_1, \cdots, \mathcal{Y}_{m_\ell} \rangle$ as

$$\widetilde{\mathcal{Y}} = \langle (\mathcal{Y}_1, 0, \cdots, 0), \cdots, (0, \cdots, \mathcal{Y}_j, \cdots, 0), \cdots, (0, \cdots, 0, \mathcal{Y}_{m_\ell}) \rangle$$

where $\widetilde{\mathcal{Y}}_j$ has $\mathcal{Y}_j$ in its j-th entry and zero strings otherwise, for every $j \in [m_\ell]$. Therefore, $\widetilde{\mathcal{Y}}$ has the same next-block-pseudo-entropy (Definition 2.14 on page 39) as $\mathcal{Y}$ and $\mathbf{H}(\widetilde{\mathcal{Y}}_j) = \mathbf{H}^{(j)}(\mathcal{Y}_j)$. That is, every block of $\widetilde{\mathcal{Y}}_j$ has $(T' - \mathrm{O}(m_\ell t), t^2(\ell\varepsilon' + 2^{-\kappa} + 2^{-ct}))$-next-block-pseudo-*min*-entropy α_t. Then, we can evaluate and concatenate the single hash function $\mathbf{H} \in_R S^{\alpha_t - \kappa}_{tm_\ell}$ on every block of $\widetilde{\mathcal{Y}} \in (\{0,1\}^{tm_\ell})^{m_\ell}$, to extract $\alpha_t - \kappa$ almost uniform bits from next-block-min-entropy α_t.

$$\widetilde{\mathsf{Ext}}\left(\widetilde{\mathcal{Y}}, \mathbf{H}\right) = \left\langle \mathbf{H}, \mathbf{H}(\widetilde{\mathcal{Y}}_1), \cdots, \mathbf{H}(\widetilde{\mathcal{Y}}_{m_\ell}) \right\rangle = \left\langle \mathbf{H}, \mathbf{H}^{(1)}(\mathcal{Y}_1), \cdots, \mathbf{H}^{(m_\ell)}(\mathcal{Y}_{m_\ell}) \right\rangle$$

It is $\left(T' - (m_\ell t)^{\mathrm{O}(1)}, m_\ell \left(t^2(\ell\varepsilon' + 2^{-\kappa} + 2^{-ct}) + 2^{-\kappa/2}\right)\right)$-pseudorandom (cf. Claim 5.4 in [HRV10], also Lemma 3.9 on page 63). The output of $\widehat{\mathsf{Ext}}$ is exactly a randomized encoding of $\widetilde{\mathsf{Ext}}$, thus it is also $\left(T' - (m_\ell t)^{\mathrm{O}(1)}, m_\ell \left(t^2(\ell\varepsilon' + 2^{-\kappa} + 2^{-ct}) + 2^{-\kappa/2}\right)\right)$-pseudorandom.

Putting above four steps together, we give the construction of G as follows

$$\begin{aligned}
& G\left(\mathbf{J}^{(1)}, \cdots, \mathbf{J}^{(t)}, \mathcal{U}_n^{(1)}, \cdots, \mathcal{U}_n^{(t\ell)}; \mathbf{H}; \mathbf{R}\right) \\
=& G_1\big(EQ\big(\mathbf{J}^{(1)}, G_{nb}(\mathcal{U}_n)^{(1)}, \cdots, G_{nb}(\mathcal{U}_n)^{(\ell)}\big), \cdots, \\
& \cdots, EQ\big(\mathbf{J}^{(t)}, G_{nb}(\mathcal{U}_n^{((t-1)\ell+1)}), \cdots, G_{nb}(\mathcal{U}_n^{(t\ell)})\big); \mathbf{H}; \mathbf{R}\big) \\
=& G_1\left(\mathcal{X}^{(1)}, \cdots, \mathcal{X}^{(t)}; \mathbf{H}; \mathbf{R}\right) \\
=& \widehat{\mathsf{Ext}}\left(\mathcal{Y}, \mathbf{H}; \mathbf{R}\right) \\
=& \Big\langle \mathbf{H}, \mathcal{X}_1^{(1)}\mathbf{H}_1^{(1)} + \mathbf{R}_1^{(1)}, \cdots, \mathcal{X}_{m_\ell}^{(1)}\mathbf{H}_1^{(m_\ell)} + \mathbf{R}_1^{(m_\ell)}, \\
& \cdots, \mathcal{X}_1^{(i)}\mathbf{H}_i^{(1)} + \mathbf{R}_{i-1}^{(1)} + \mathbf{R}_i^{(1)}, \cdots, \mathcal{X}_{m_\ell}^{(i)}\mathbf{H}_i^{(m_\ell)} + \mathbf{R}_{i-1}^{(m_\ell)} + \mathbf{R}_i^{(m_\ell)}, \\
& \cdots, \mathcal{X}_1^{(t)}\mathbf{H}_t^{(1)} + \mathbf{R}_{t-1}^{(1)}, \cdots, \mathcal{X}_{m_\ell}^{(t)}\mathbf{H}_t^{(m_\ell)} + \mathbf{R}_{t-1}^{(m_\ell)} \Big\rangle
\end{aligned}$$

G has input length $t(\log m + n\ell) + m_\ell t(\alpha_t - \kappa) + m_\ell(t-1)(\alpha_t - \kappa)$, and output

length $m_\ell t(\alpha_t - \kappa) + m_\ell t(\alpha_t - \kappa)$. To get stretch, it suffices to have

$$\begin{aligned}
&t(\log m + n\ell) - m_\ell(\alpha_t - \kappa) \\
=&t(\log m + n\ell) - (\ell - 1)m\Big(t \cdot \frac{k}{m} - O\big(\log t \cdot \sqrt{t \cdot \kappa}\big) - \kappa\Big) \\
=&t(\log m + n\ell) - (\ell - 1)t(n + \log n) + (\ell - 1)m\Big(O\big(\log t \cdot \sqrt{t \cdot \kappa}\big) + \kappa\Big) \\
=&t \log m + t(n + \log n) - \ell t \log n + (\ell - 1)m\Big(O\big(\log t \cdot \sqrt{t \cdot \kappa}\big) + \kappa\Big) \\
<&0
\end{aligned}$$

This holds for sufficiently large n when $\ell = n/\log n$, $t = \mathrm{O}\left(n^2 \log^2 n\right)$, $\kappa = \log^2 n$ and $m = \mathrm{O}\left(n\right)$. Moreover, G is $\Big(T' - (m_\ell t)^{\mathrm{O}(1)}, m_\ell \left(t^2(\ell\varepsilon' + 2^{-\kappa} + 2^{-ct}) + 2^{-\kappa/2}\right)\Big)$-pseudorandom, which turns out $\Big(T' - n^{\mathrm{O}(1)}, n^{\mathrm{O}(1)}\big(\varepsilon' + 2^{-\log^2 n/2}\big)\Big)$-pseudorandom according to the choice of ℓ, t, κ. Therefore G is a pseudorandom generator if G_{nb} is a next-block-pseudo-entropy generator as in Lemma 2.15 (page 41). □

Note that we use a family of linear hash functions because it is not clear how to implement with streaming algorithms the description-succinct hash family in [VZ12, HRV10]. This causes loss in efficiency (which contrasts the purpose of improving seed efficiency in [VZ12, HRV10]), but here we strive for a streaming feasibility result which is not at all obvious how to get.

2.5 Streaming Public-Key Encryption

We construct an IND-CPA (defined on page 26) secure PKE system based on [Reg05] the LWE assumption together with the PKE construction, where the encryption and the decryption algorithms are streaming algorithms, henceforth called *streaming PKE*. The key generation is not a streaming algorithm and the private keys contain a good deal of redundancy. We also show that large private keys are necessary. The lower bound on the length of the private key stated in Theorem 4.8 (p. 105), and discussed in Section 4.2.2.

Theorem 2.16. *Given the decision-LWE assumption (Assumption 2.18), the construction in Section 2.5.1 is an IND-CPA secure PKE. Moreover, both the encryption and the decryption algorithms are streaming computable.*

The main challenge of a streaming PKE is the decryption algorithm. The techniques we developed so far do not apply, because the decryption algorithm should output exactly the plaintext rather than any code.

We construct our streaming PKE based on the decision-LWE assumption. The intuition of such assumption is exposited in [Reg05], which also gives reductions from worst-case lattice problems (at the time of the writing of this dissertation, these lattice assumptions and reductions are used in common places).

Definition 2.17 (LWE problem)**.** *Let $q = q(n) \leq \mathsf{poly}(n)$, consider a list of equations $b_i = \langle \mathbf{s}, \mathbf{a}_i \rangle + e_i \pmod q$ for $i = 1, 2, \cdots, \mathsf{poly}(n)$, where $\mathbf{s} \in \mathbb{Z}_q^n$, $\mathbf{a}_i \in_R \mathbb{Z}_q^n$ and $b_i \in \mathbb{Z}_q$. If furthermore $e_i \in \mathbb{Z}_q$ follows a discrete Gaussian distribution*[7] *with parameter α, we denote by* search-$\mathsf{LWE}_{q,\alpha}$ *the problem of recovering $\mathbf{s}$ from such equations. In* decision-$\mathsf{LWE}_{q,\alpha}$ *the goal is to distinguish $(\mathbf{a}, \langle \mathbf{s}, \mathbf{a} \rangle + e \mod q)$ from $\mathcal{U}_{\mathbb{Z}_q^{n+1}}$ with non-negligible advantage, when both $\mathbf{s}, \mathbf{a} \in_R \mathbb{Z}_q^n$.*

Assumption 2.18 (cf. [Reg05, MP12])**.** *When $\alpha \geq 2\sqrt{n}$, search-$\mathsf{LWE}_{q,\alpha}$ cannot be solved in probabilistic polynomial time with non-negligible probability. If $\alpha \geq \omega(\sqrt{n} \log n)$ then decision-$\mathsf{LWE}_{q,\alpha}$ cannot be solved in probabilistic polynomial time with non-negligible advantage.*

2.5.1 The Construction

In our construction the public and private keys are "streaming usable" forms of the following two matrices: $\mathbf{A}$ and a random matrix $\mathbf{D}$. Matrix $\mathbf{A}$ is statistically close to uniform, and at the same time orthogonal to $\begin{bmatrix} \mathbf{I} \\ \mathbf{D} \end{bmatrix}$. The latter consists of short vectors which cannot be retrieved from a uniformly random matrix. This is also

[7] A discrete Gaussian distribution over $\mathbb{Z}_q$ is defined by $D_{\mathbb{Z}_q,\alpha}(x) = \rho_\alpha(x/q)/\rho_\alpha(\mathbb{Z}_q)$, where $\rho_\alpha(x) = \sum_{k=-\infty}^{\infty} \alpha^{-1} \exp(-\pi(\frac{x+k}{\alpha})^2)$ follows a continuous Gaussian distribution, and $\rho_\alpha(\mathbb{Z}_q) = \sum_{x \in \mathbb{Z}_q} \rho_\alpha(x/q)$.

known as the lattice hardness assumption, which is popular nowadays due to its computational simplicity and resistant to attacks by quantum computers.

KeyGen: Pick a matrix $\mathbf{D} \in \mathbb{Z}_p^{(m-w)\times w}$ uniformly at random from $\{0, \pm 1\}^{(m-w)\times w}$. Uniformly at random pick $\overline{\mathbf{A}} \in \mathbb{Z}_q^{n\times(m-w)}$, and compute $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ as $\mathbf{A} = [-\overline{\mathbf{A}}\mathbf{D} \mid \overline{\mathbf{A}}] \mod q$. Let $\begin{bmatrix} \mathbf{I} \\ \mathbf{D} \end{bmatrix} = [\mathbf{d}_1, \cdots, \mathbf{d}_w]$. Here $k = \lceil 2 \log n \rceil, q = 2^k, m = 3nk, w = nk$, for the security parameter n.

For an arbitrary message length N, **KeyGen** outputs N copies of $\mathbf{A}$ as the public key, and nN copies of $\mathbf{d}_1$ as the private key. Each copy of $\mathbf{A}$ is written in row-first order, i.e. $(a_{11}, a_{12}, \cdots, a_{1m},\ a_{21}, \cdots, a_{2m}, \cdots,\ a_{n1}, \cdots, a_{nm})$.

Enc: On input a message $\mathbf{x} = (x^{(1)}, \cdots, x^{(N)}) \in \{0,1\}^N$, for $i = 1, 2, \cdots, N$, uniformly choose $\mathbf{s}_i \in_R \mathbb{Z}_q^n$ and $\mathbf{x}_i \in_R \left\{qx^{(i)}/2\right\} \times \mathbb{Z}_q^{w-1}$.

Sample $\mathbf{e}_i \in \mathbb{Z}_q^m$ for $i = 1, 2, \cdots, N$, where each entry $e_{ij} \sim \mathcal{D}_\alpha$ follows the discrete Gauss distribution with mean 0 and standard deviation α, for $j = 1, 2, \cdots, m$.

For every $i = 1, 2, \cdots, N$, sequentially output y_i, where y_i is a randomized encoding of $\mathbf{s}_i^T\mathbf{A} + \mathbf{e}_i^T + (\mathbf{x}_i^T, \mathbf{0}) \mod q$. That is, for $\mathbf{R} \in_R \mathbb{Z}_q^{(n-1)\times m}$, realizing $\mathbf{e}_i^T, (\mathbf{x}_i^T, \mathbf{0})$ as $1 \times m$ row vectors, and recalling that $\mathbf{A}$ is an $n \times m$ matrix, we define y_i is the row-first order of $\mathbf{Y}_i$ as follows

$$\mathbf{Y}_i = \begin{bmatrix} s_{i1} & & \\ & \ddots & \\ & & s_{in} \end{bmatrix} \cdot \mathbf{A} + \begin{bmatrix} \mathbf{R} \\ (\mathbf{x}^T, \mathbf{0}) \end{bmatrix} + \begin{bmatrix} \mathbf{e}_i^T \\ -\mathbf{R} \end{bmatrix}$$

Dec: Given the ciphertext $\{y_i\}_{i=1,2,\cdots,N}$ and the decryption key nN copies of $\mathbf{d}_1$. We recover the message $\mathbf{x} \in \{0,1\}^N$ by computing $b = [1\ 1\ \cdots\ 1]_{1\times n}\mathbf{Y}_i\mathbf{d}_1 \mod q$ and outputing $x^{(i)} = \lfloor 2b/q + 1/2 \rfloor \mod 2$ for every $i = 1, \cdots, N$.

Comparison with [Reg05]. The above construction is similar to the PKE construction in [Reg05]. We borrow from [MP12] the key generation and encryption algorithms which enable us to turn them into streaming computable encryption/decryption.

Note that [MP12], unlike the above construction, achieves a CCA-secure PKE (defined on page 50). By the time when the dissertation is written, we do not know how to perform ciphertext validity checks (as in e.g. [MP12]) in a streaming fashion.

2.5.2 Correctness, Efficiency, and Security Analysis

We show correctness, IND-CPA security, and how to encrypt/decrypt in a streaming fashion.

Correctness. Since $\mathbf{A}\begin{bmatrix}\mathbf{I}\\ \mathbf{D}\end{bmatrix} = \mathbf{0}$ by the construction of $\mathbf{A}$, and $[1\ 1\ \cdots\ 1]_{1\times n}\mathbf{Y}_i = \mathbf{s}_i^t\mathbf{A} + \mathbf{e}_i^t + (\mathbf{x}_i^t, \mathbf{0}) \mod q$:

$$[1\ 1\ \cdots\ 1]_{1\times n}\mathbf{Y}_i \begin{bmatrix}\mathbf{I}\\ \mathbf{D}\end{bmatrix} = \mathbf{e}_i^t \begin{bmatrix}\mathbf{I}\\ \mathbf{D}\end{bmatrix} + \mathbf{x}_i^t \mod q$$

In particular, the first entry of above vector is $b = [1\ 1\ \cdots\ 1]_{1\times n}\mathbf{Y}_i\mathbf{d}_1 = \mathbf{e}_i^t\mathbf{d}_1 + x^{(i)}\cdot q/2 \mod q$. As the summation of $\|\mathbf{d}_1\|$ samples from $\mathcal{D}_\alpha$, $\mathbf{e}_i^t\mathbf{d}_1$ follows the Gaussian distribution $\mathcal{D}_{\sqrt{\|\mathbf{d}_1\|}\cdot\alpha}$ where $\|\mathbf{d}_1\| \le O(n\log n)$. Therefore, $\Pr[|\mathbf{e}_i^t\mathbf{d}_1| > \sqrt{n\log n}\alpha] < 2^{-\Omega(n)}$. As long as $\alpha < \frac{n^2}{\sqrt{n\log n}} < \frac{q}{\sqrt{n\log n}}$, the noise $\mathbf{e}_i^t\mathbf{d}_1$ is bounded by $q/4$ with overwhelming probability $1 - 2^{-\Omega(n)}$. That is, $x^{(i)}$ can be determined with error less than $2^{-\Omega(n)}$. Recalling that the assumption only requires α to be $\omega(\sqrt{n}\log n)$, we can set $\alpha = \sqrt{n}(\log n)^{3/2}$.

Streaming computability. For the encryption scheme, the input includes the plaintext $\mathbf{x}$ together with N copies of $\mathbf{A}$ in row-first order and a stream of random bits $(\mathbf{r^x}_1, \mathbf{r^e}_1, \cdots, \mathbf{r^x}_N, \mathbf{r^e}_N, \mathbf{r}_1, \cdots, \mathbf{r}_N)$, where each $\mathbf{r^x}_i$ contains the randomness for $\mathbf{x}_i$ and $\mathbf{r^e}_i$ for sampling $\mathbf{e}_i$, and $\mathbf{r}_i = (r_{11}^{(i)}, r_{12}^{(i)}, \cdots, r_{(n-1)m}^{(i)})$ denotes the random bits used to encode y_i. With a single pass we generate sequentially $\mathbf{x}_1^t, \mathbf{e}_1^t, \cdots, \mathbf{x}_N^t, \mathbf{e}_N^t$. By an identical construction as in the proof of Theorem 2.12 we can copy random bits $\mathbf{r}^{(i)}$ and compute all y_i's in parallel with logarithmic memory and constant passes over two streams. Thus, the encryption is computable in the streaming model.

In the decryption scheme, for each i we first compute the column vector $\mathbf{Y}_i\mathbf{d}_1$, then add all its entries to determine b. However, since $\mathbf{Y}_i$ appears in row-first order, b is not computable by reading a single copy of $\mathbf{d}_1$ with constant passes. We use multiple copies of $\mathbf{d}_1$ in the private key to compute in a streaming fashion $\mathbf{Y}_i\mathbf{d}_1$.

Our key generation algorithm **KeyGen** cannot be implemented as a streaming algorithm, since it involves a matrix multiplication $\overline{\mathbf{A}}\mathbf{D}$, and generates polynomially many copies of $\mathbf{A}$ and $\mathbf{d}_1$. This is somehow acceptable in the sense that **KeyGen** invokes only once at the beginning.

We conclude by discussing the length-efficiency of the construction in Section 2.5.1. For an N-bit long plaintext, the ciphertext contains $N \times nm$ elements from $\mathbb{Z}_q$, which has length $Nmnk = O(Nn^2 \log^2 n)$, while both the public and the private key are of the same length. The amount of random bits used in **Enc** is also asymptotically the same. This is because for every i, $\mathbf{r}_i^x$ and $\mathbf{r}_i$ together require $mnk - 1$ many random bits, and $\mathbf{r}_i^e$ is bounded by $O(mn)$. In conclusion, a factor of $O(n^2 \log^2 n)$ is introduced to the length of ciphertext and keys. The blow-up in the key length is somehow inevitable, as we show a lower bound linear in N on the length of the private key.

Security. The above constructed PKE system is IND-CPA secure.

Claim 2.19. *If Assumption 2.18 is true, then the construction in Section 2.5.1 is IND-CPA secure.*

Proof. Let us consider the standard IND-CPA security experiment. First, the adversary is given the public key $\mathbf{A}$ and he chooses two distinct message $\mathbf{x}, \mathbf{x}' \in \{0,1\}^N$ within polynomial time. Then, the challenger computes two ciphertexts $\mathbf{y}, \mathbf{y}' \in \mathbb{Z}_q^{N \times mn}$ using private random bits. For IND-CPA security, it suffices to prove that $\mathbf{y}$ and $\mathbf{y}'$ are computationally indistinguishable.

To prove $\mathbf{A}$ is statistically close to uniform, we introduce the following lemma (c.f. Lemma 4.5.1 in [HILL99], also in [ILL89]).

Lemma 2.20 (Leftover Hash Lemma)**.** *Suppose that a random variable* $\mathcal{X} \in \{0,1\}^n$ *has min-entropy at least* m *and* $\ell = m - 2\epsilon$*. Then,*

$$\delta\big(\langle H_n^\ell(\mathcal{X}), H_n^\ell\rangle, \langle \mathcal{U}_\ell, H_n^\ell\ \rangle\big) \leq 2^{-(\epsilon+1)}$$

where $\delta(\cdot)$ *denotes the statistical distance function, and the two appearances of* H_n^ℓ *in* $\langle H_n^\ell(\mathcal{X}), H_n^\ell\rangle$ *refer to the same sample.*

Since $\overline{\mathbf{A}}$ is uniformly chosen, $\mathbf{D}$ has min-entropy $(\log_2 3)(m-w)w$, and the binary representation of $-\overline{\mathbf{A}}\mathbf{D}$ is of length $nw\log q = nwk < (\log_2 3)(m-w)w - 2nwk$. We identify the uniformly random matrix $\overline{\mathbf{A}}$ as a linear hash function which shrinks $\mathbf{D}$ with min-entropy $(\log_2 3)(m-w)w$ to less than $(\log_2 3)(m-w)w - 2nwk$ bits. Then,

$$\delta\big(\langle -\overline{\mathbf{A}}\mathbf{D}, \overline{\mathbf{A}}\rangle, \langle \mathcal{U}_{\mathbb{Z}_q^{n\times w}}, \mathcal{U}_{\mathbb{Z}_q^{n\times(m-w)}}\rangle\big) \leq 2^{-(nwk+1)} < 2^{-n^2\log^2(n)}$$

Therefore, the distribution of $\mathbf{A} = [-\overline{\mathbf{A}}\mathbf{D}|\overline{\mathbf{A}}]$ has negligible statistical distance from the uniform distribution. Conditioning this event we assume $\mathbf{A}$ as uniformly chosen though it incurs small loss.

We show that if our PKE is not secure then Assumption 2.18 is not true. Suppose the PKE system is not IND-CPA secure: an adversary algorithm $\mathcal{A}$, who knows $\mathbf{A}$ but not $\mathbf{s}$, distinguishes $\mathbf{y}$ from $\mathbf{y}'$ with non-negligible advantage, i.e. $|\Pr[\mathcal{A}(\mathbf{y}) = 1] - \Pr[\mathcal{A}(\mathbf{y}') = 1]| \geq n^{-O(1)}$ over the randomness of $\mathbf{A}$ and internal random coins in the encryption. Then applying the standard hybrid argument $\mathcal{A}$ implies a distinguisher $\mathcal{A}^*$ for $y_i \neq y_i'$ with non-negligible advantage. In particular, $\mathcal{A}^*$ must be able to distinguish the first column of $\mathbf{Y}_i$ and $\mathbf{Y}_i'$, for other columns are identically distributed. $\mathcal{A}^*$ essentially distinguishes between $(\mathbf{a}_1, \mathbf{s}_i^t\mathbf{a}_1 + e_{i1} + q/2 \pmod q)$ and $(\mathbf{a}_1, \mathbf{s}_i^t\mathbf{a}_1 + e_{i1} \pmod q)$ which are information theoretically equivalent to y_i and y_i', recalling the construction of **Enc** and the fact that y_i is a randomized encoding of $\mathbf{s}_i^t\mathbf{A} + \mathbf{e}_i^t + (\mathbf{x}_i^t, \mathbf{0}) \pmod q$. In the meanwhile, $\mathcal{A}^*$ requires only $m-1$ samples $(\mathbf{a}_j, \mathbf{s}_i^t\mathbf{a}_j + e_{ij} \mod q)$ for $j = 2, 3, \cdots, m$ (while y_j in $\mathbf{y}$ does not help at all as long as $j \neq i$, since $\mathcal{A}^*$ can sample them internally).

Now, for randomly chosen $\mathbf{s}_i \in_R \mathbb{Z}_q^n$, $\mathcal{A}^*$ distinguishes $(\mathbf{a}_i, \langle \mathbf{s}_i, \mathbf{a}_i \rangle + e_i \mod q)$ (or equivalently $(\mathbf{a}_i, \langle \mathbf{s}_i, \mathbf{a}_i \rangle + e_i + q/2 \mod q)$) from $(\mathbf{a}_i, \mathcal{U}_{\mathbb{Z}_q})$ with non-negligible advantage. Moreover, $\mathcal{A}^*$ has polynomial running time and uses at most $m - 1 = \mathsf{poly}(n)$ samples $(\mathbf{a}_j, \langle \mathbf{s}, \mathbf{a}_j \rangle + e_j \mod q)$ for $\mathbf{a}_j \in_R \mathbb{Z}_q^n$ and $j = 2, 3, \cdots, m$.

However, such $\mathcal{A}^*$ solves exactly the decision-$\mathsf{LWE}_{q,\alpha}$ problem with $\alpha = \sqrt{n}(\log n)^{3/2}$, which immediately contradicts Assumption 2.18. Thus, in conclusion, the PKE construction in Section 2.5.1 is IND-CPA secure. □

2.5.3 Remarks on CCA-Security in the Multi-Stream Model

A PKE system is secure against *Chosen Ciphertext Attack* (CCA) if no polynomial time randomized adversary can win the following security experiment with probability greater than $1/\mathsf{poly}(n)$. Such a PKE is also called a *CCA-secure PKE*.

i. The challenger runs **KeyGen** and uses its random choices to generate a public $\mathcal{PK}$ and a private $\mathcal{SK}$ key, and reveals the $\mathcal{PK}$ to the adversary.

ii. The adversary on input 1^n and $\mathcal{PK}$ chooses ciphertexts $c_1, \cdots, c_m$ for $m = \mathsf{poly}(n)$, and sends them to the challenger.

iii. The challenger computes $x_i = \mathbf{Dec}(\mathcal{SK}, c_i)$ for every i, and sends $x_1, \ldots, x_m$ to the adversary.

iv. The challenger selects $x \sim \mathcal{U}_n$ and sends $c = \mathbf{Enc}(\mathcal{PK}, x)$ to the adversary.

v. The adversary wins if it outputs $x' = x$ based on $\mathcal{PK}$, $x_1, \ldots, x_m$, and c.

Our IND-CPA secure construction is similar to the CCA-secure PKE in [MP12]. At this stage we cannot conjecture either way about the existence of CCA-secure PKE in the multi-stream model. In what follows we discuss difficulties in achieving CCA-security with streaming decryption, even when the encryption is a polynomial time algorithm.

A weak point of a streaming decryption is that it lacks the ability of performing common types of validity checks on the ciphertext. Due to entropy-memory size considerations with $\mathrm{O}(1)$ many passes a streaming algorithm can perform at most $\mathrm{O}(\log n)$ many independent tests when checking whether a given input is legal. This

gives potential advantage to an adversary who can sample polynomially many ciphertexts.

In particular, as far as it concerns the CCA-secure scheme of [MP12], we first note that the way the LWE assumption is used is by exactly making a validity test on the ciphertext (see previous paragraph). Furthermore, the [MP12] construction involves operations we cannot compute in a streaming sense (matrix multiplication, inverting a random matrix). It is not clear how these operations can be adapted in a streaming fashion. Note that when decrypting a ciphertext there is not much flexibility. In a correct decryption we always get the unique encrypted message, whereas for encryption there is no such thing as "correct ciphertext" which allows one to use e.g. randomized encodings. We discuss in this section about the limitations of streaming PKE systems. It is somehow inevitable that our construction achieves merely IND-CPA secure (as in [Reg05]) starting from the CCA-security in [MP12]. A major weakness point is that a streaming decryption algorithm lacks the power of performing validity check. Since the streaming decryption algorithm is able to perform at most $\mathrm{O}(\log n)$ independent tests to check whether the given input is a legal ciphertext. By randomly picking polynomially many ciphertexts, the adversary collects polynomially many uniformly distributed legal ciphertexts, and it may extract a lot of information from those legal ciphertexts.

Another weakness point comes from the usage of LWE assumption. By such assumption, modifying the error vector in a small range would not change the decrypted message, while the decrypted message varies for big changes in the error vector. Therefore by observing the decrypted message for differentially changed error vectors, it seems quite hopeful to find out the real value without masked by any error vector. As soon as the error vector is eliminated, the learning problem becomes much easy to solve.

We further note that many important operations in the [MP12] construction are difficult for a streaming device. For example, inverting a random matrix and multiplying two matrices. That is why we failed in adapting the [MP12] construction for the streaming model. Moreover, we strongly suspect the existence of any CCA-secure

PKE with constant-pass streaming decryption schemes. There are two reasons why we cannot follow the same argument: the first is the lack of validity checking methods in the streaming model, which gives the adversary too much power to extract information with chosen ciphertext; the second reason appears to be the difficulty of matrix operations in their decryption algorithm, such as inversion and multiplication.

2.6 Conclusions and Remarks on Practicality

Our work leaves open the possibility of streaming cryptography for a number of popular private and public-key primitives. As a next step we propose to study the streaming possibility for the following cryptographic primitives: (i) linear-stretch pseudorandom generators, (ii) CCA-secure PKE, (iii) signature schemes, and (iv) message authentication.

It is also open whether the number of passes we achieve (see Table 2.2 below) are optimal, and also simultaneously improve the seed-efficiency of streaming pseudorandom generators from NC^1 one-way functions. For example, our generic streaming one-way function is done with 5 passes, whereas when starting from a concrete assumption (Assumption 2.21) we can do it with 4 passes, which is optimal.

	# of passes	**external tapes**
one-way function	5	1 Read-Only (RO) & 1 Read-Write (RW)
pseudorandom generator	7	2 RW
	15	1 RO & 1 RW
PKE **Enc**	5	1 RO & 1 RW
PKE **Dec**	2	2 RO *

Table 2.2: **Expense of cryptographic primitives in the multi-stream model.** * The private key and ciphertext are given in different external tapes.

Some remarks on practicality. Randomized encodings generally demand huge amounts of randomness (typically $\Omega(n^4)$) for input length n, and thus our generic compilers can be understood as feasibility results. In practice, starting from concrete

intractability assumptions we can do much better. Here is a practical example which in fact resembles a lot the one in [AIK10] (but a few model-specific differences – our model is not two dimensional but things are arranged similarly).

Assumption 2.21 (Decoding Random Linear Codes (DRLC))**.** *A* random linear code f_{code} *maps* $f_{code} : (\mathbf{A}, \mathbf{x}, \mathbf{e}) \mapsto (\mathbf{A}, \mathbf{Ax} + \mathbf{e})$*, where* $\mathbf{A} \in \mathsf{GF}(2)^{m \times n}$*,* $\mathbf{x} \in \mathsf{GF}(2)^n$*,* $\mathbf{e} \in \mathsf{GF}(2)^m$*. Choose positive constants* κ, ϵ, δ *such that* $\kappa = \frac{n}{m} < 1 - H_2\big((1+\epsilon)\delta\big)$*, where* $H_2(p) = -p \log_2 p - (1-p)\log_2(1-p)$ *for* $p < 1/2$ *and* $H_2(p) = 1$ *otherwise. If* $\mathbf{A}, \mathbf{x}$ *are chosen uniformly at random, while* $\mathbf{e}$ *has at most* $\frac{\delta m}{2}$ *one-entries, then* f_{code} *is a one-way function.*

Theorem 2.22. *Suppose that the DRLC assumption holds true. Then, there exists a one-way function* F *computable by a streaming algorithm with* 2 *streams,* 4 *passes and* $O(\log n)$ *internal memory. Furthermore, if the DRLC input is of size* N *the corresponding input size for* F *is* $n \leq 2N$*.*

Construction outline. Suppose the random bits $(r_{11}, r_{21}, \cdots, r_{mn})$ are given on the extra stream (this is without loss of generality/not necessary), and parse the input stream as $(x_1, a_{11}, a_{21}, \cdots, a_{m1}, \cdots, x_n, a_{1n}, \cdots, a_{mn}, e_1, \cdots, e_m)$.

First, by scanning over both two streams, we compute $(a_{11}x_1 + r_{11}, \cdots, a_{m1}x_1 + r_{m1}, a_{12}x_2 + r_{12}, \cdots, a_{m2}x_2 + r_{m2}, \cdots, a_{1n}x_n + r_{1n}, \cdots, a_{mn}x_n + r_{mn}, e_1, \cdots, e_m)$.

Then, we compute $(a_{11}x_1 + r_{11}, \cdots, a_{m1}x_1 + r_{m1}, a_{12}x_2 + r_{12} - r_{11}, \cdots, a_{m2}x_2 + r_{m2} - r_{m1}, \cdots, a_{1n}x_n + r_{1n} - r_{1(n-1)}, \cdots, a_{mn}x_n + r_{mn} - r_{m(n-1)}, e_1 - r_{1n}, \cdots, e_m - r_{mn})$. This is the randomized encoding of $\mathbf{Ax} + \mathbf{e}$ computable with 4 passes over 2 streams. □

We end this chapter with a note on the practicality of the multi-stream model. One physical analog of a stream is a hard-disk or a disk-array. It makes sense to think of physical disks of linear size, e.g. $2n$ or $3n$, for any input length n, as long as the input fits in the disks. Whereas, a polynomial blow-up, e.g. n^2 or n^3, is not reasonable when n is already quite large (even comparable to the size of disks). For pragmatic analysis of computation in the multi-stream model, we believe that this *stream-size* parameter should be added to the other parameters: number of passes,

local memory size and number of streams. In this dissertation, all constructions make very few passes and the stream size never exceeds $2 \times \mathsf{input\ length}$.

Chapter 3

Randomness Extraction in the Multi-Stream Model

In this chapter, we discuss randomness extraction in the multi-stream model and from real-world Big Data. We first provide an overview of the problem and previous works, then present the construction and mathematical analysis of the RRB extractor in the multi-stream model. Finally, we describe the experiments of real-world randomness extraction in Section 3.3, with the experimental results summarized in Section 3.4.

By Definition 1.5 (page 14), a *seeded randomness extractor* is a function $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ that takes two inputs. The first input is a sample from the source and the second input is the seed. We say that Ext *is a* (k,ε)*-extractor* if for every (n,k)-source X and a uniformly random seed $Y \sim \mathcal{U}_d$, the output distribution $\mathsf{Ext}(X,Y)$ is ε-close to uniform, i.e. $\mathcal{SD}\left(\mathsf{Ext}(X,Y),\mathcal{U}_m\right) \leq \varepsilon$. In typical settings, $d = \mathsf{polylog}(n)$, $\varepsilon = 1/\mathsf{poly}(n)$, $k = n^{\Omega(1)}$, and $m = \Omega\left(k\right)$ or at least $m = k^{\Omega(1)}$.

For this type of extractors, there are non-explicit constructions ([Sha11, NZ96], by probabilistic method) with optimal parameters, and explicit constructions, e.g. Trevisan's extractor [Tre99] and its followups, that achieved nearly optimal parameters, i.e. for every $\varepsilon > 0$, $k = n^{\Omega(1)}$ and any constant $\delta > 0$, they extract $m = k^{1-\delta}$ bits with seed length $d = \mathrm{O}\left(\log n\right)$.

For specific sources X, rather than the worst-case (n,k)-source chosen by an adversary, extracting randomness becomes considerably easier. For example, if all

the bits in X are independent and identically distributed (not necessarily uniform), then uniform random bits can be generated following a simple strategy – partition X into pairs of bits, for each pair of bits output the first if and only if they are distinct. This simple algorithm is known as von Neumann extractor [von51] and easy to prove its correctness for the specific source X. In this work, we consider the following specific sources.

Definition 3.1 (Bit-fixing sources)**.** *An* (n,k)-source X *is an* (n,k)-bit-fixing source*, if there exists and* $T \subseteq [n]$*,* $|T| = k$*, and* $y \in \{0,1\}^{n-k}$*, such that* $X_T \sim \mathcal{U}_k$ *while* $X_{[n]\setminus T} = y$ *is a fixed constant (string).*

Note that every bit-fixing source is in particular an affine source.

Definition 3.2 (Affine sources)**.** *We say that* X *is an* (n,k)-affine source *if* X *is uniformly distributed over a* k*-dimensional affine subspace of* $\{0,1\}^n$*. That is,* $X \stackrel{def}{=} X' \cdot \mathbf{A} + \mathbf{b}$ *for a fixed matrix* $\mathbf{A} \in \{0,1\}^{k\times n}$ *and row vector* $\mathbf{b} \in \{0,1\}^n$*, and a* k*-dimensional random row vector* $X' \sim \mathcal{U}_k$*.*

We say that Ext is a (k,ε)-*extractor for bit-fixing sources* (resp. *affine sources*) if $\mathcal{SD}\left(\mathsf{Ext}(X,\mathcal{U}_d),\mathcal{U}_m\right) \leq \varepsilon$ holds for every (n,k)-bit-fixing source (resp. (n,k)-affine source) X. Although there are deterministic extractors for bit-fixing sources [KZ06, Rao09b, GRS06], it is unclear whether there is extractors for bit-fixing sources in the multi-stream model. Furthermore, the known lower bound results (e.g. Theorem 4.9 and Theorem 4.11 in Section 4.3, also [BYRST02]) are proved for bit-fixing sources.

Another variant is the *multi-source extractor*, which takes as input multiple samples from independent sources. Multi-source extractors are able to extract from general weak sources without a uniform random seed [BIW06, DEOR04, Raz05, Zuc90], although they are mostly theoretical achievements and not suitable for practical applications on big objects. We implement a seedless multi-source extractor (cf. [BIW06, Zuc90]) for the initial seed generation in our experimental study (Section 3.3.2 on page 83).

For randomness extractors in the multi-stream model, we consider super-constant passes since in Theorem 4.9 (on page 109) we obtain an $\Omega\left(\log\log n\right)$ lower bound.

Formally, a *streaming extractor* is an $(o(\log n), O(\log n), O(1))$-streaming algorithm where (i) the sample from the source is written on the first stream in the beginning of the computation, and (ii) the random seed of size $\mathsf{polylog}(n)$ is stored in the local memory. For *multi-source streaming extractors*, we concatenate all input samples in the same input stream. In the discussion about lower bounds (see Section 4.3 on page 108), we allow free access to the seed (regardless of size), which makes the lower bounds stronger.

3.1 The Random Re-Bucketing (RRB) Extractor

Here we present the RRB extractor together with its streaming implementation. This extractor makes $O\left(\log\log\frac{n}{\varepsilon}\right)$ many passes. We also present its first provable property, which is that it extracts randomness from bit-fixing sources. RRB is a parametric algorithm. In the next section we will see that with different parameters RRB enjoys next-block-entropy guarantees and hence works for more general sources.

At a high-level, RRB consists of three stages, as depicted in Figure 3-1.

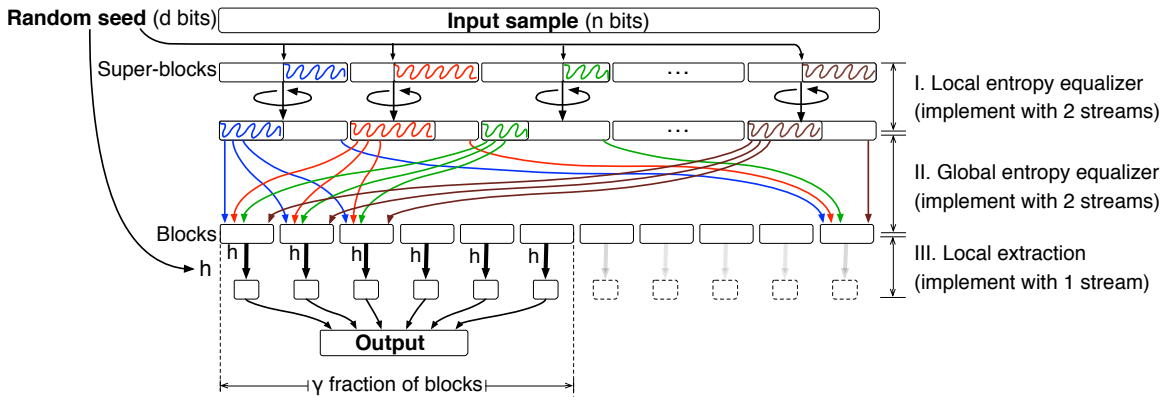

Figure 3-1: **The Random Re-Bucketing (RRB) extractor.** The random seed of size $\mathsf{polylog}(n)$ is used only in Stages I & III. In Stage III the same local extractor h is used for the first γ fraction of blocks. The number of super-blocks b also depends on an error tolerance ε and the empirically estimated min-entropy rate κ. In the main body, we explain how to realize this description as an algorithm that uses *two streams*.

I. Partition the n-bit long input into $b = \mathrm{O}\left(\log_2 \frac{n}{\varepsilon}\right)$ many *super-blocks*, each of length n/b. Inside each super-block, choose uniformly and independently a random point to cyclically shift the super-block.

II. Re-bucket the b *super-blocks* into n/b many *blocks* each of size b, where the i-th block consists of the i-th bit from every super-block, for $i = 1, 2, \ldots, n/b$.

III. Specify a *local extractor* $h : \{0,1\}^b \to \{0,1\}^{\kappa b/2}$ using the uniform random seed; for example, h can be a random Toeplitz matrix. Then, locally apply h on the first $b_O = \gamma n/b$ blocks, concatenate, and output the result. Here the *effectiveness factor* $\gamma = \Omega(1)$ denotes the fraction of blocks used for local extraction.

Note that only stage I and III use randomness, whereas stage II performs a fixed but "streaming-friendly" permutation. Therefore, the seed y has following structure:

$$y = \langle \underbrace{h}_{\text{hash function}}, \underbrace{y_1, \ldots, y_b}_{\text{indices for cyclic shifts}} \rangle$$

The final local extraction can be performed in-place since the blocks are of size $b = \mathrm{O}\left(\log \frac{n}{\varepsilon}\right)$ which is sufficiently small.

Let us now introduce the shifting operator and after that we will describe RRB.

Definition 3.3. *For $z = (z_1, \ldots, z_m) \in \{0,1\}^m$ and $r \in \{0, 1, 2, \ldots, m-1\}$, $\mathsf{shift}(z, r)$ denotes the string that obtained by cyclic shifting z to the left for r-bits, i.e. $\mathsf{shift}(z, r) = (z_{r+1}, z_{r+2}, \ldots, z_m, z_1, z_2, \ldots, z_r)$.*

For input length n, min-entropy $k = \Omega(n)$, and error tolerance $\varepsilon > 0$, the construction $\mathsf{RRB} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ has parameters $m = k - \mathrm{o}(k)$, $b = \Theta(\log \frac{n}{\varepsilon})$ and $d = b + b \log \frac{n}{b}$, $b_O = n/b$. For convenience and without loss of generality, we further assume b to be a power of 2. Let σ_1, σ_2 denote the two streams and let $\sigma_2[j]$ denote the j-th bit in σ_2. Perhaps it helps to think of these parameters as follows.

Typical values

input length	n	output length	$m = \Omega(n)$
error tolerance	$\varepsilon = 1/n^{\Omega(1)}$	number of super-blocks	$b = \mathrm{O}(\log n)$
min-entropy	$k = \Omega(n)$	seed length	$d = \mathrm{O}\left(\log^2 n\right)$

Here is the pseudo-code description of RRB.

Input: $\sigma_1 \leftarrow x$, where x is an n-bit-long sample from weak source

Result: an m-bit-long binary string distributed ε-close to $\mathcal{U}_m$

Initialization:

$k \leftarrow$ lower bound on the min-entropy of the source

$b \leftarrow$ the number of super-blocks

$d \leftarrow$ seed length

$y \leftarrow \mathcal{U}_d$ the random seed

partition $y = (h, y_1, \ldots, y_b) \in \{0,1\}^{2b} \times \left\{0, 1, 2, \ldots, \frac{n}{b} - 1\right\}^b$,

where h describes a hash function $h : \{0,1\}^b \to \{0,1\}^{m/b_O}$

Process:

1 partition $\sigma_1 = x = \left(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_b\right)$, where $|\overline{x}_i| = \frac{n}{b}$ for $1 \le i \le b$

2 $\sigma_2 \leftarrow (\mathsf{shift}(\overline{x}_1, y_1), \ldots, \mathsf{shift}(\overline{x}_b, y_b))$

3 **for** $j = 1$ **to** $\left\lceil \log b \right\rceil$ **do**

4 $\quad \sigma_1 \leftarrow \left(\sigma_2[1], \sigma_2[\frac{n}{2} + 1], \sigma_2[2], \sigma_2[\frac{n}{2} + 2], \ldots, \sigma_2[\frac{n}{2}], \sigma_2[n]\right)$

5 $\quad \sigma_2 \leftarrow \sigma_1$

6 **end**

7 partition $\sigma_1 = \left(z_1, \ldots, z_{n/b}\right)$ where $|z_i| = b$ for $1 \le i \le \frac{n}{b}$

8 **Output:** $\sigma_2 \leftarrow \left(h(z_1), \ldots, h(z_{b_O})\right)$

Algorithm 2: RRB – randomness extractor in the multi-stream model

Claim 3.4. RRB *is streaming computable with* $\mathrm{O}(\log b)$ *passes over two streams and* $\mathrm{O}(b + \log n)$ *local memory.*

Proof. We analyze the number of passes and the usage of local memory stage-by-stage.

In Stage I (steps 1 and 2), step 2 makes four passes by copying the corresponding substrings (i.e. the super-blocks) from σ_1 to σ_2. In particular, in two passes write to σ_2 the left part of $\mathsf{shift}(\overline{x}_i, y_i)$ together with placeholders of the right part (i.e. $(\overline{x}_{i,y_i+1}, \overline{x}_{i,y_i+2}, \ldots, \overline{x}_{i,n/b}, *, ..., *)$) for $i \in [b]$; then invoke another two passes to fill in the placeholders with the rest of the y_i bits of $\mathsf{shift}(\overline{x}_i, y_i)$ (i.e. $\overline{x}_{i,1}, \ldots, \overline{x}_{i,y_i}$). Note, the latter does not need to store every y_i in memory, since y_i is already recorded implicitly as the length of the placeholders.

In Stage II (the for-loop, steps 3–6), the iteration costs $\mathrm{O}(\log b)$ passes in total since the inner loop requires constant many passes: step 4 takes two passes over σ_1 and one pass over σ_2; step 5 is trivial and in fact it can be done by simply renaming σ_1 and σ_2 without any actual process over the streams.

Stage III (steps 7 and 8) is done using two passes in total. For every $i \in \left\{1, 2, \ldots, \frac{n}{b}\right\}$ it reads z_i from σ_1 into memory and then write $h(z_i)$ to σ_2. Since h is computable within $\mathrm{O}(\log n)$ space, $b + \mathrm{O}(\log n)$ local memory is sufficient to buffer z_i and compute $h(z_i)$.

Therefore, RRB uses $\mathrm{O}(\log b)$ passes over two streams σ_1, σ_2 and $\mathrm{O}(b + \log n)$ local memory. □

3.1.1 Streaming Extraction from Bit-fixing Sources

In the following theorem, we validate that RRB works on (n, k)-bit-fixing sources.

Theorem 3.5. *For every $k = \Omega(n)$, there exists $b = \Theta(\log \frac{n}{\varepsilon})$, such that for $d = b + b \log \frac{n}{b}$ and $b_O = \frac{n}{b}$, RRB is a (k, ε)-extractor for every (n, k)-bit-fixing source with $m = k - \mathrm{o}(k)$.*

Proof. The proof relies on the following two lemmas. Lemma 3.6 asserts that the first two stages of RRB achieves next-block-min-entropy guarantee. Lemma 3.9 illustrates how the last stage of RRB extracts randomness from next-block-min-entropy.

Lemma 3.6. *Fix an arbitrary (n,k)-bit-fixing source X, $Y \sim \mathcal{U}_d$, and let random variable*[1] *$Z = (Z_1, \ldots, Z_{n/b})$ be the content of σ_1 in step 5 in the computation of $\mathsf{RRB}(X,Y)$. Then, for every $k = \Omega(n)$ and for every positive $\delta < \frac{k}{n}$, with probability greater than $1 - \frac{n}{b} \cdot \exp(-2\delta^2 b) = 1 - 2^{-\Omega(b)}n$ over the random choice of Y, all blocks in Z have next-block-min-entropy $\alpha = \left(\frac{k}{n} - \delta\right) b$.*

Proof of Lemma 3.6. We first analyze the structure of Z and then bound the probability that z_j has min-entropy $\Omega(kb/n)$. We conclude by union bound the next-block-min-entropy lower bound for the blocks in Z.

Since X is an (n,k)-bit-fixing source without loss of generality there exists $S \subseteq [n]$ consisting of k indices such that $X_S = \mathcal{U}_k$, whereas $X_{[n]\setminus S}$ is fixed. Let $S_j = S \bigcap \left\{\frac{n}{b}(j-1)+1, \frac{n}{b}(j-1)+2, \ldots, \frac{n}{b}\cdot j\right\}$ denote the indices of unfixed bits in the j-th input block for $j \in [b]$ (i.e. $\overline{x}_j$ in step 1 of RRB). Then, let $k_j = |S_j|$ and $k = \sum_{i=j}^{b} k_j$.

Now, we upper bound $\Pr[\mathsf{H}_\infty[Z_1] < \alpha]$, the probability over the random choice of Y that Z_1 does not have min-entropy $\alpha = \left(\frac{k}{n} - \delta\right) b$. Note that for every fixed Y, Z_1 is a deterministic projection of $X = (X_1, \ldots, X_n)$. Thus, its min-entropy equals to the number of unfixed bits from X. Therefore, it suffices to count the number of unfixed bits in Z_1.

For every $Y = y$ and y_j determined by y as in RRB, let $I_j = I_j(y_j)$ be the indicator that the j-th bit of Z_1 is not fixed. However, as long as b is a power of 2 the j-th bit of Z_1 is also the first bit in $\mathsf{shift}(\overline{X}_j, y_j)$, i.e. $Z_{1,j} = \mathsf{shift}(\overline{X}_j, y_j)_1 = (\overline{X}_j)_{y_j+1} = X_{\frac{n}{b}(j-1)+y_j+1}$. Thus, $\mathsf{H}_\infty[Z_1] = \sum_{j=1}^{b} I_j$ where for every $j \in [b]$, $I_j = 1$ if and only if $\frac{n}{b}(j-1) + y_j + 1 \in S$.

By definition, y_j's are independent and uniformly chosen from $\left\{0, 1, \ldots, \frac{n}{b} - 1\right\}$. Thus, (i) all I_j's are independent from each other; (ii) $\frac{n}{b}(j-1) + y_j + 1$ uniformly distributes in $\left\{\frac{n}{b}(j-1)+1, \frac{n}{b}(j-1)+2, \ldots, \frac{n}{b}\cdot j\right\}$ and hence $I_j = 1$ if and only if $\frac{n}{b}(j-1) + y_j + 1 \in S_j$. Then, by definition of k_j we have $\mathrm{E}[I_j] = k_j b/n$ for $j \in [b]$,

[1]Here, we abuse notation and we write Z instead of z because we wish to refer to the transformation of the algorithm on the statistical source. For the same reason we use $\overline{X}_j$ in the argument instead of $\overline{x}_j$ in the description of RRB to emphasis it as a random variable depending on X.

and furthermore, $\mathrm{E}\left[\sum_{j=1}^{b} I_j\right] = \sum_{j=1}^{b} \mathrm{E}\left[I_j\right] = \sum_{j=1}^{b} \frac{k_j b}{n} = \frac{kb}{n}$.

Since $\mathsf{H}_\infty[Z_1] = \sum_{j=1}^{b} I_j$, $\mathrm{E}\left[\mathsf{H}_\infty[Z_1]\right] = \frac{kb}{n}$ and $I_1, \ldots, I_b$ are independent we upper bound $\Pr\left[\mathsf{H}_\infty[Z_1] < \alpha\right]$ by Hoeffding's inequality (see below)

$$\begin{aligned}
&\Pr\left[\mathsf{H}_\infty[Z_1] < \alpha\right] \\
&= \Pr\left[\sum_{j=1}^{b} I_j < \alpha\right] = \Pr\left[\sum_{i=j}^{b} I_j < \mathrm{E}\left[\sum_{j=1}^{b} I_j\right] - \delta b\right] \\
&\leq \exp\left(-\frac{2(\delta b)^2}{b}\right) = \exp\left(-2\delta^2 b\right)
\end{aligned}$$

Lemma 3.7 (Hoeffding's Inequality)**.** *For independent almost surely bounded random variables $X_1, \ldots, X_n$, i.e. $\Pr[X_i \in [a_i, b_i]] = 1$ for $i \in [n]$, let $Y = X_1 + \cdots + X_n$. Then, for any positive Δ,*

$$\Pr\left[Y - \mathrm{E}\left[Y\right] \geq \Delta\right] \leq \exp\left(-\frac{2\Delta^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right)$$

Since $\Pr\left[\mathsf{H}_\infty[Z_1] < \alpha\right] \leq \exp\left(-2\delta^2 b\right)$ and Z_1 is symmetric[2] to $Z_2, \ldots, Z_{n/b}$, by union bound he have that

$$\Pr\left[\exists j \in [\frac{n}{b}] \text{ such that } \mathsf{H}_\infty[Z_j] < \alpha\right] \leq \frac{n}{b} \cdot \exp\left(-2\delta^2 b\right) = 2^{-\Omega(b)} \cdot \frac{n}{b}$$

Noticing that Z is a permutation of X and X is a bit-fixing source, we have that $\mathsf{H}_\infty[Z_j \mid Z_1, Z_2, \ldots, Z_{j-1}] = \mathsf{H}_\infty[Z_j]$ for every $j \in [\frac{n}{b}]$. The above inequality is equivalent to

$$\Pr\left[\forall j \in [\frac{n}{b}],\ \mathsf{H}_\infty[Z_j \mid Z_1, \ldots, Z_{j-1}] \geq \alpha = \left(\frac{k}{n} - \delta\right) b\right] \geq 1 - \frac{n}{b} \cdot \exp\left(-2\delta^2 b\right)$$

□

We introduce the Leftover Hash Lemma [ILL89, IZ89] and apply it in the proof of Lemma 3.9, which asserts that a single extractor with the same seed suffices to

[2]Unlike other sources that lack such a degree of independence, bit-fixing sources are technically easier to handle exactly because the random bits are independent and this induces symmetry for all Z_i's.

extract randomness from each of the blocks.

Lemma 3.8 (Leftover Hash Lemma[ILL89, IZ89])**.** *Let $S \subseteq \{0,1\}^n$ and $|S| \geq 2^\alpha$. Let $\rho > 0$ and $\mathcal{H}$ be a family of 2-universal hash functions mapping n bits to $\alpha - \rho$ bits. Then, the distribution $(h, h(x))$ is at most $2^{-\rho/2}$-close to uniform, when h is chosen uniformly at random from $\mathcal{H}$ and x sampled uniformly from S.*

Lemma 3.9 (cf. Lemma 6 in [Zuc96])**.** *Let $\mathcal{H}$ be a family of 2-universal hash functions mapping b bits to $\alpha-\rho$ bits. If a random variable $Z = (Z_1, \ldots, Z_\ell)$ has next-block-min-entropy α and h is chosen uniformly at random from $\mathcal{H}$, then $(h, h(Z_1), \ldots, h(Z_\ell))$ is $2^{-\rho/2} \cdot \ell$ close to uniform distribution.*

Proof of Lemma 3.9. We use the standard hybrid argument and for $i \in \{0, 1, \ldots, \ell\}$ we let $H_i(Z) \stackrel{\text{def}}{=} \big(h, h(Z_1), \ldots, h(Z_i), \mathcal{U}_{(\alpha-\rho)(\ell-i)}\big)$. In particular, $H_0(Z) = \mathcal{U}_{|h|+(\alpha-\rho)\ell}$ and $H_\ell(Z) = (h, h(Z_1), \ldots, h(Z_\ell))$. Thus, it suffices to bound $\mathcal{SD}\left(H_\ell(Z), H_0(Z)\right)$.

Since $Z = (Z_1, \ldots, Z_\ell)$ has next-block-min-entropy α it follows that

$$\mathsf{H}_\infty\left[Z_i \;\middle|\; h(Z_1), \ldots, h(Z_{i-1})\right] \geq \mathsf{H}_\infty\left[Z_i \;\middle|\; Z_1, \ldots, Z_{i-1}\right] \geq \alpha$$

Therefore, by the Leftover Hash Lemma (page 63), $\mathcal{SD}\left(H_i(Z), H_{i-1}(Z)\right) \leq 2^{-\rho/2}$ holds for every $i \in \{1, \ldots, \ell\}$. Finally, we have

$$\mathcal{SD}\left(H_\ell(Z), H_0(Z)\right) \leq \sum_{i=1}^{\ell} \mathcal{SD}\left(H_i(Z), H_{i-1}(Z)\right) \leq 2^{-\rho/2} \cdot \ell$$

□

The universal family of hash function in Lemma 3.9 can be uniformly sampled with $b+\alpha-\rho-1 \leq 2b$ bits. For concreteness define $h : \{0,1\}^b \to \{0,1\}^{\alpha-\rho}$ as follows

$$h(\mathbf{z}) \stackrel{\text{def}}{=} \mathbf{T}\mathbf{z}$$

where $\mathbf{T}$ is a randomly chosen $(\alpha-\rho) \times b$ Toeplitz matrix (first appeared on page 32), i.e. by uniformly sampling $b+\alpha-\rho-1$ elements from $\mathbb{Z}_2$, and all operations are over

$\mathbb{Z}_2$. Note that there exist hashing families that can be sampled with b-many bits but not practically realizable for big sources. This is why we use the Toeplitz hashing family here.

By Lemma 3.6 and Lemma 3.9 and by setting $\ell = b_O = \frac{n}{b}$, we conclude that with probability at least $1 - \frac{n}{b} \cdot \exp\left(-2\delta^2 b\right)$ over the random choice of Y, the statistical distance is bounded as $\mathcal{SD}\left(\mathsf{RRB}(X,Y), \mathcal{U}_{(\alpha-\rho)b_O}\right) \leq 2^{-\rho/2} \cdot \frac{n}{b}$. That is, for X and $Y \sim \mathcal{U}_d$ as above, and for $m = (\alpha-\rho)\ell = \left(\left(\frac{k}{n} - \delta\right) b - \rho\right) \frac{n}{b} = k - \delta n - \frac{\rho n}{b}$,

$$\mathcal{SD}\left(\mathsf{RRB}(X,Y), \mathcal{U}_{(\alpha-\rho)b_O}\right) \leq \left(2^{-\rho/2} + \exp\left(-2\delta^2 b\right)\right) \cdot \frac{n}{b}$$

Consequently, RRB is a (k,ε)-extractor for $k = \Omega(n)$, $\varepsilon = \left(2^{-\rho/2} + \exp\left(-2\delta^2 b\right)\right) \cdot \frac{n}{b}$ and $m = k - \left(\delta + \frac{\rho}{b}\right) n$. For every constant $\delta > 0$, it suffices to set $\rho = 2\log\frac{n}{\varepsilon}$ and $b = \frac{1}{2\delta^2} \ln\frac{n}{\varepsilon} = \Theta(\log\frac{n}{\varepsilon})$, such that $m = k - (\delta + 4\delta^2 \log e)n$. Therefore, RRB is a (k,ε)-extractor for bit-fixing sources with $m = k - \mathrm{o}(k)$ and $b = \Theta(\log\frac{n}{\varepsilon})$ as long as $k = \Omega(n)$.

Moreover, RRB is streaming computable with $\mathrm{O}\left(\log\frac{n}{\varepsilon}\right)$ memory and $\mathrm{O}(\log b) = \mathrm{O}\left(\log\log\frac{n}{\varepsilon}\right)$ passes over two streams by Claim 3.4. □

3.2 Extraction from Affine and General Sources

Stages I and II of RRB provably guarantee a strong lower bound on the next-block-entropy. We show this in Section 3.2.1. If instead of guaranteed next-block-entropy we had next-block-min-entropy then by Lemma 3.9 Stage III would have extracted (almost) uniform random bits. Section 3.2.2 and Section 3.2.3 show how the next-block-entropy guarantee implies a next-block-min-entropy guarantee for affine sources and general sources. In Section 3.2.4 we present the random extractor for general sources. then again we obtain a next-block-min-entropy guarantee.

3.2.1 The RRB Next-Block-Entropy Guarantee

We show that at the end of Stage II in RRB the input X is transformed into Z and Z has guaranteed next-block-entropy whenever $\mathsf{H}_\infty[X] = \Omega(n)$.

Lemma 3.10. *Let X be a random variable over $\{0,1\}^n$ such that $\mathsf{H}_\infty[X] \geq k = cn$ for a constant $c > 0$. Let $Y \sim \mathcal{U}_d$ and $Z = (Z_1, \ldots, Z_{n/b})$ at step 5 in the computation of $\mathsf{RRB}(X, Y)$. Then, for every positive constant $c_0 < 1 - (1-c)^{2/3}$ and for every $\ell < \frac{n\left(1-(1-c)^{2/3}-c_0\right)}{b^2}$, each of $Z_1, \ldots, Z_\ell$ has next-block-entropy $\Omega(b)$ with probability at least $1 - 2^{-\Omega(b)}$ over the random choice of Y.*

Proof. In the construction of RRB (page 57), Stages I & II operate as follows:
(I) partition X into b super-blocks $(X_1, \ldots, X_b)$ and perform cyclic shifting inside each super-block $(\mathsf{shift}(X_1, Y_1), \ldots, \mathsf{shift}(X_b, Y_b))$;
(II) convert b super-blocks to n/b blocks by re-bucketing, such that $Z = \left(Z_1, \ldots, Z_{n/b}\right)$ $= \left((\mathsf{shift}(X_1, Y_1)_1, \ldots, \mathsf{shift}(X_b, Y_b)_1), \ldots\ldots, \left(\mathsf{shift}(X_1, Y_1)_{n/b}, \ldots, \mathsf{shift}(X_b, Y_b)_{n/b}\right)\right)$.

We assert that many super-blocks in X have next-block-entropy $\Omega(n/b)$. For every constant $c_1 > c$, there exist $S \subseteq [b]$ such that $|S| \geq \frac{(c_1-1)cb}{c_1-c} \cdot b$ and for every $i \in S$, $\mathsf{H}\left[X_i \mid X_1, \ldots, X_{i-1}\right] \geq \frac{k}{c_1 b}$. Otherwise, suppose there exists S satisfying $|S| < \frac{(c_1-1)c}{c_1-c} b$ and $\mathsf{H}\left[X_i \mid X_1, \ldots, X_{i-1}\right] < \frac{k}{c_1 b}$ for every $i \in [b] \backslash S$. Then,

$$\begin{aligned}
\mathsf{H}[X] =& \sum_{i=1}^{b} \mathsf{H}\left[X_i \mid X_1, \ldots, X_{i-1}\right] \\
=& \sum_{i \in S} \mathsf{H}\left[X_i \mid X_1, \ldots, X_{i-1}\right] + \sum_{i \in [b]\backslash S} \mathsf{H}\left[X_i \mid X_1, \ldots, X_{i-1}\right] \\
<& \sum_{i \in S} |X_i| + \sum_{i \in [b]\backslash S} \frac{k}{c_1 b} \\
=& |S| \cdot \frac{n}{b} + (b - |S|)\frac{k}{c_1 b} = \frac{k}{c_1} + \frac{n}{b}\left(1 - \frac{c}{c_1}\right) \cdot |S| < k
\end{aligned} \tag{3.1}$$

Now, we show $\mathsf{H}\left[Z_j \mid Z_1, Z_2, \ldots Z_{j-1}\right] \geq \Omega(b)$ with probability $1 - 2^{-\Omega(b)}$ (over the choice of Y) for every $j \in \{1, \ldots, \ell\}$. Recall that $Z_j[i]$ denotes the i-th bit in Z_j for every $i \in [b]$. It suffices to lower bound $\mathsf{H}\left[Z_j[i] \mid Z_1, \ldots, Z_{j-1}; Z_j[1], \ldots, Z_j[i-1]\right]$ for every $i \in S$ and sum them up.

$$
\begin{aligned}
&\mathsf{H}\left[Z_j[i] \;\middle|\; Z_1,\ldots,Z_{j-1}; Z_j[1],\ldots,Z_j[i-1]\right]\\
=&\mathsf{H}\left[Z_j[i] \;\middle|\; (Z_1[1],\ldots,Z_1[b]),\ldots,(Z_{j-1}[1],\ldots,Z_{j-1}[b]); (Z_j[1],\ldots,Z_j[i-1])\right]\\
=&\mathsf{H}\big[Z_j[i] \;\big|\; (Z_1[1],\ldots,Z_j[1]),\ldots,(Z_1[i-1],\ldots,Z_j[i-1]);\\
&\qquad (Z_1[i],\ldots,Z_{j-1}[i]),\ldots,(Z_1[b],\ldots,Z_{j-1}[b])\big]\\
\geq&\mathsf{H}[Z_j[i] \;\big|\; (Z_1[1],\ldots,Z_{n/b}[1]),\ldots,(Z_1[i-1],\ldots,Z_{n/b}[i-1]);\\
&\qquad (Z_1[i],\ldots,Z_{j-1}[i]),\ldots,(Z_1[b],\ldots,Z_{j-1}[b])\big]\\
=&\mathsf{H}\big[Z_j[i] \;\big|\; \mathsf{shift}(\overline{X}_1,Y_1),\ldots,\mathsf{shift}(\overline{X}_{i-1},Y_{i-1});\\
&\qquad (Z_1[i],\ldots,Z_{j-1}[i]),\ldots,(Z_1[b],\ldots,Z_{j-1}[b])\big]\\
\geq&\mathsf{H}\left[Z_j[i] \;\middle|\; (\overline{X}_1,Y_1),\ldots,(\overline{X}_{i-1},Y_{i-1}); (Z_1[i],\ldots,Z_{j-1}[i]),\ldots,(Z_1[b],\ldots,Z_{j-1}[b])\right]\\
=&\mathsf{H}\left[Z_j[i] \;\middle|\; \overline{X}_1,\ldots,\overline{X}_{i-1}; (Z_1[i],\ldots,Z_{j-1}[i]),\ldots,(Z_1[b],\ldots,Z_{j-1}[b])\right] \qquad (3.2)
\end{aligned}
$$

In the above (3.2) holds because for every $i \in [b]$, $Z_j[i] = \mathsf{shift}(\overline{X}_i, Y_i)_j = \overline{X}_{i,Y_i+j}$ is independent from $Y_1,\ldots,Y_{i-1}$.

Since $Z_j[i]$ is drawn from $\overline{X}_i$ let us first lower bound the conditional entropy of $\overline{X}_i$. Recalling that for every $i \in S$, $\mathsf{H}\left[\overline{X}_i \;\middle|\; \overline{X}_1,\ldots,\overline{X}_{i-1}\right] \geq \frac{k}{c_1 b}$,

$$
\begin{aligned}
&\mathsf{H}\left[\overline{X}_i \;\middle|\; \overline{X}_1,\ldots,\overline{X}_{i-1}; (Z_1[i+1],\ldots,Z_{j-1}[i+1]),\ldots,(Z_1[b],\ldots,Z_{j-1}[b])\right]\\
\geq&\mathsf{H}\left[\overline{X}_i \;\middle|\; \overline{X}_1,\ldots,\overline{X}_{i-1}\right] - \mathsf{H}\left[(Z_1[i+1],\ldots,Z_{j-1}[i+1]),\ldots,(Z_1[b],\ldots,Z_{j-1}[b])\right]\\
\geq&\mathsf{H}\left[\overline{X}_i \;\middle|\; \overline{X}_1,\ldots,\overline{X}_{i-1}\right] - (j-1)(b-i)\\
>&\frac{k}{c_1 b} - (j-1)(b-i) > \frac{k}{c_1 b} - jb
\end{aligned}
$$

For notational simplicity we let $k_1 \stackrel{\text{def}}{=} \frac{k}{c_1 b} - jb$ and

$$
C_i \stackrel{\text{def}}{=} \left(\overline{X}_1,\ldots,\overline{X}_{i-1}; (Z_1[i+1],\ldots,Z_{j-1}[i+1]),\ldots,(Z_1[b],\ldots,Z_{j-1}[b])\right)
$$

Similarly to (3.1) we prove that for $S' \stackrel{\text{def}}{=} \left\{i' \in [\tfrac{n}{b}] \middle| \mathsf{H}\left[\overline{X}_{i,i'} \;\middle|\; C_i; (\overline{X}_{i,1},\ldots,\overline{X}_{i,i'-1})\right] \geq k_2\right\}$

there must be $|S'| > \frac{k_1 - k_2 n/b}{1-k_2}$ since otherwise

$$\begin{aligned}\mathsf{H}\left[\overline{X}_i \mid C_i\right] &= \sum_{i'=1}^{n/b} \mathsf{H}\left[\overline{X}_{i,i'} \mid C_i; (\overline{X}_{i,1}, \ldots, \overline{X}_{i,i'-1})\right] \\ &\leq 1 \cdot |S'| + k_2 \cdot \left(\frac{n}{b} - |S'|\right) \leq k_1\end{aligned}$$

Recalling (3.2) and for $Y_i = y_i$ such that $y_i + j \in S'$ we have

$$\begin{aligned}&\mathsf{H}\left[Z_j[i] \mid Z_1, \ldots, Z_{j-1}; Z_j[1], \ldots, Z_j[i-1]\right] \\ \geq &\mathsf{H}\left[Z_j[i] \mid \overline{X}_1, \ldots, \overline{X}_{i-1}; (Z_1[i], \ldots, Z_{j-1}[i]), \ldots, (Z_1[b], \ldots, Z_{j-1}[b])\right] \\ = &\mathsf{H}\left[Z_j[i] \mid C_i; (Z_1[i], \ldots, Z_{j-1}[i])\right] \\ = &\mathsf{H}\left[\overline{X}_{i,y_i+j} \mid C_i; (\overline{X}_{i,y_i+1}, \ldots, \overline{X}_{i,y_i+j-1})\right] \\ \geq &\mathsf{H}\left[\overline{X}_{i,y_i+j} \mid C_i; (\overline{X}_{i,1}, \ldots, \overline{X}_{i,y_i+j-1})\right] \geq k_2 \qquad (3.3)\end{aligned}$$

Recalling that Y_i is uniformly chosen from $\left\{0, 1, \ldots, \frac{n}{b} - 1\right\}$ for every $j \in [\frac{n}{b}]$ there is $\Pr[Y_i + j \in S'] \geq \frac{|S'|-j+1}{n/b}$. Let $p \stackrel{\text{def}}{=} \frac{|S'|-j+1}{n/b}$. Then,

$$\mathrm{E}\left[\mathsf{H}\left[Z_j[i] \mid Z_1, \ldots, Z_{j-1}; Z_j[1], \ldots, Z_j[i-1]\right]\right] \geq pk_2$$

New notation. Let $\mathsf{H}_S \stackrel{\text{def}}{=} \sum_{i \in S} \mathsf{H}\left[Z_j[i] \mid Z_1, \ldots, Z_{j-1}; Z_j[1], \ldots, Z_j[i-1]\right]$.

Now, apply Lemma 3.7 to get the following bound, where the probability is taken over $r_1, \ldots, r_b$ used for shifting.

$$\Pr\left[\mathsf{H}_S < pk_2|S| - \delta\right] \leq \Pr\left[\mathsf{H}_S < \mathrm{E}\left[\mathsf{H}_S\right] - \delta\right] \leq \exp\left(-\frac{2\delta^2}{|S|}\right)$$

Since $\mathsf{H}\left[Z_j \mid Z_1, \ldots, Z_{j-1}\right] \geq \sum_{i \in S} \mathsf{H}\left[Z_j[i] \mid Z_1, \ldots, Z_{j-1}; Z_j[1], \ldots, Z_j[i-1]\right] = \mathsf{H}_S$, we conclude

$$\Pr\left[\mathsf{H}\left[Z_j \mid Z_1, \ldots, Z_{j-1}\right] < pk_2|S| - \delta\right] \leq \exp\left(-\frac{2\delta^2}{|S|}\right) \qquad (3.4)$$

Finally, we specify the parameters. Recall that $k_1 = \frac{k}{c_1 b} - jb$ and set c_1, k_2 as

follows,

$$c_1 = \frac{c}{1-(1-c)^{2/3}}, k_2 = 1 - \sqrt{1 - \frac{k_1 b}{n}} = 1 - \sqrt{(1-c)^{2/3} + \frac{jb^2}{n}}$$

Then, by the way we define S, S' and p we have

$$\begin{aligned} b \geq &|S| \geq \frac{(c_1-1)c}{c_1 - c} \cdot b = \left(1-(1-c)^{1/3}\right) b \\ \frac{n}{b} \geq &|S|' > \frac{k_1 - k_2 n/b}{1-k_2} = \frac{n}{b}\left(\frac{k_1 b/n - k_2}{1-k_2}\right) = \frac{n}{b}\left(1 - \sqrt{(1-c)^{2/3} + \frac{jb^2}{n}}\right) \\ p = &\frac{|S|' - j + 1}{n/b} > \left(1 - \sqrt{(1-c)^{2/3} + \frac{jb^2}{n}}\right) - \frac{jb}{n} \end{aligned}$$

and in addition $1 - \exp\left(-\frac{2\delta^2}{b}\right) \leq 1 - \exp\left(-\frac{2\delta^2}{|S|}\right)$.

Thus, with probability at least $1 - \exp\left(-\frac{2\delta^2}{b}\right)$ we have

$$\begin{aligned} &\mathsf{H}\left[Z_j \mid Z_1, \ldots, Z_{j-1}\right] \geq pk_2|S| - \delta \\ &> \left(1 - \sqrt{(1-c)^{2/3} + \frac{jb^2}{n}} - \frac{jb}{n}\right)\left(1 - \sqrt{(1-c)^{2/3} + \frac{jb^2}{n}}\right)\left(1-(1-c)^{1/3}\right) b - \delta \end{aligned}$$

Recalling that $j \leq \ell < \frac{n\left(1-(1-c)^{2/3}-c_0\right)}{b^2}$ for $c_0 > 0$ and thus $\frac{jb^2}{n} + (1-c)^{2/3} < 1 - c_0$ and moreover $1 - \sqrt{(1-c)^{2/3} + \frac{jb^2}{n}} - \frac{jb}{n} = \Omega(1)$, there exists $\delta = \Theta(b)$ such that

$$\mathsf{H}\left[Z_j \mid Z_1, \ldots, Z_{j-1}\right] > \Omega(1) \cdot b - \delta = \Omega(b)$$

We conclude that Z_j has next-block-entropy $\Omega(b)$ with probability at least $1 - \exp\left(-2\delta^2/b\right) = 1 - 2^{-\Omega(b)}$.

By union bound the first ℓ blocks in Z, i.e. $Z_1, \ldots, Z_\ell$, each has next-block-entropy $\Omega(b)$ with probability at least $1 - \exp\left(-2\delta^2/b\right)\ell = 1 - 2^{-\Omega(b)}\ell$, for every $\ell < \frac{n\left(1-(1-c)^{2/3}-c_0\right)}{b^2} = \Omega\left(\frac{n}{b^2}\right)$. □

By Lemma 3.10 we immediately obtain the following corollary.

Corollary 3.11. *For every (n,k)-source X with $k = \Omega(n)$, there exists $b = \mathrm{O}\left(\log \frac{n}{\varepsilon}\right)$ and $\ell = \Omega\left(\frac{n}{\log^2 \frac{n}{\varepsilon}}\right)$, such that each of $Z_1, \ldots, Z_\ell$ has next-block-entropy $\Omega\left(\log \frac{n}{\varepsilon}\right)$ with probability $1 - \frac{\varepsilon}{n}$.*

3.2.2 Streaming Extraction from Affine Sources

We show that for specific parameters the RRB extractor also works for affine sources.

Theorem 3.12. *For every $\varepsilon > 0$ and $k = \Omega(n)$, there exists $b = \Theta(\log \frac{n}{\varepsilon}), d = \mathrm{O}(b \log n), \ell = \Omega\left(\frac{n}{b^2}\right)$, and $m = \Omega\left(\frac{n}{b}\right)$ such that $\mathsf{RRB} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k,ε)-extractor for every (n,k)-affine source.*

The proof of Theorem 3.12 consists of two parts. First, Lemma 3.10 guarantees certain next-block-entropy, and then Lemma 3.13 states that for affine sources next-block-entropy also means next-block-min-entropy.

Lemma 3.13. *For every (n,k)-affine source $X = X' \cdot \mathbf{A} + \mathbf{b}$, and every partition $X = (X_1, X_2)$, $\mathsf{H}_\infty\left[X_2 \mid X_1\right] = \mathsf{H}\left[X_2 \mid X_1\right]$.*

Proof. Let $\mathbf{A} = [\mathbf{A_1}\ \mathbf{A_2}]$ and $\mathbf{b} = (\mathbf{b_1}, \mathbf{b_2})$ such that

$$X_1 = X' \cdot \mathbf{A_1} + \mathbf{b_1},\ X_2 = X' \cdot \mathbf{A_2} + \mathbf{b_2}$$

By the definition of conditional entropy and denote the rank of a matrix by $\mathsf{rank}(\cdot)$, we have

$$\mathsf{H}\left[X_2 \mid X_1\right] = \mathsf{H}[X] - \mathsf{H}[X_1] = \mathsf{rank}(\mathbf{A}) - \mathsf{rank}(\mathbf{A_1})$$

On the other hand $\mathsf{H}_\infty\left[X_2 \mid X_1\right] = -\log\left(\max_{(y,z)\in R} \Pr[X_2 = z \mid X_1 = y]\right)$, where R denotes the range of $X'\mathbf{A}$ and the probability is taken over $X' \sim \mathcal{U}_k$. By the

definition of X_1, X_2

$$\begin{aligned}
&\Pr[X_2 = z \mid X_1 = y] \\
=&\Pr\left[X' \cdot \mathbf{A_2} + \mathbf{b_2} = z \mid X' \cdot \mathbf{A_1} + \mathbf{b_1} = y\right] \\
=&\Pr\left[X' \cdot \mathbf{A_2} + \mathbf{b_2} = z \wedge X' \cdot \mathbf{A_1} + \mathbf{b_1} = y\right] / \Pr\left[X' \cdot \mathbf{A_1} + \mathbf{b_1} = y\right] \\
=&\Pr\left[X' \cdot \mathbf{A} + \mathbf{b} = (y, z)\right] / \Pr\left[X' \cdot \mathbf{A_1} + \mathbf{b_1} = y\right] \\
=&2^{-\mathsf{rank}(\mathbf{A})} / 2^{-\mathsf{rank}(\mathbf{A_1})} = 2^{-\mathsf{rank}(\mathbf{A}) + \mathsf{rank}(\mathbf{A_1})}
\end{aligned}$$

Therefore, we conclude that

$$\begin{aligned}
\mathsf{H}_\infty\left[X_2 \mid X_1\right] =& -\log\left(\max_{(y,z)\in R} \Pr[X_2 = z \mid X_1 = y]\right) \\
=&\mathsf{rank}(\mathbf{A}) - \mathsf{rank}(\mathbf{A_1}) = \mathsf{H}\left[X_2 \mid X_1\right]
\end{aligned}$$

Thus, next-block-entropy transforms to next-block-min-entropy for affine sources. □

Note that Z in Step 5 of RRB is in particular a permutation of the input X, which is an (n, k)-affine source when X follows an (n, k)-affine source and $(y_1, \ldots, y_b)$ is fixed. As a result, Lemma 3.10 and Lemma 3.13 together imply that each of the first ℓ blocks of Z have next-block-min-entropy $b' = \Omega(b)$, with probability $1 - 2^{-b''}t$, where $b'' = \Omega(b)$. Then, by Lemma 3.9 we use $y_0 \in \{0,1\}^{2b}$ to sample $h : \{0,1\}^b \to \{0,1\}^{(b'-\rho)}$ and apply h on every block of $(Z_1, \ldots, Z_\ell)$, which has next-block-min-entropy $b' = \Omega(b)$. Hence, we obtain $(b' - \rho)t$ bits that are $2^{-\rho/2}\ell$ close to uniform.

Putting things together and by setting $\ell = \Theta(n/b^2)$ as in Lemma 3.10, $\rho = b'' \geq 2\left(\log\frac{1}{\varepsilon} + \log\ell + 1\right)$ and $b' > 2\rho$ we are ready to conclude the theorem. Specifically, $b = \Theta(\log\frac{n}{\varepsilon})$ and RRB is a (k, ε)-extractor for (n, k)-affine sources with $m = \ell(b' - \rho) \geq \ell b'/2 = \Omega(n/b)$ and statistical error at most $(2^{-b''} + 2^{-\rho/2})\ell \leq \varepsilon$.

3.2.3 Next-Block-Entropy to Next-Block-Min-Entropy

To prove that RRB extracts randomness from every (n, k)-source, we show how to convert the guaranteed next-block-entropy (Lemma 3.10 on page 65) to next-block-

min-entropy.

We denote by $X^\lambda \stackrel{\text{def}}{=} \left(X_1, \ldots, X_\lambda\right)$ the concatenation of $\lambda \in \mathbb{Z}^+$ independent and identically distributed[3] random variables $X_1, \ldots, X_\lambda$ over $\mathcal{U} = \{0,1\}^n$. For every $x \in \mathcal{U}$, let $\mathsf{H}_X(x) \stackrel{\text{def}}{=} \log \frac{1}{\Pr[X=x]}$ for $x \in \operatorname{Supp}(X)$, and let $\mathsf{H}_X(x) \stackrel{\text{def}}{=} 0$ otherwise. We also abuse notation and let $\mathsf{H}_X(x_i) \stackrel{\text{def}}{=} \log \frac{1}{\Pr[X_i=x_i]}$, when there is no ambiguity.

If every random variable X_i has entropy $\mathsf{H}[X_i] = \mathrm{E}_{x_i \sim X_i}\left[\mathsf{H}_X(x_i)\right] \geq \alpha$, then there exists random variable $\widetilde{X}$, such that $\mathcal{SD}\left(\widetilde{X}, X^\lambda\right) \leq \varepsilon$ and $\widetilde{X}$ has min-entropy $\mathsf{H}_\infty[\widetilde{X}] \geq \lambda\alpha - \mathrm{O}\left(\sqrt{\lambda \cdot \log(1/\varepsilon)} \cdot \log\left(|\mathcal{U}| \cdot \lambda\right)\right)$.

Theorem 3.14 (cf. Lemma 2.1 in [HRV10])**.** *Let $\lambda \in \mathbb{Z}^+$ and $X_1, \ldots, X_\lambda$ be independent random variables taking values in $\mathcal{U} = \{0,1\}^n$ with (Shannon) entropy $\mathsf{H}[X_i] \geq \alpha$ for every $i \in \{1, \ldots, \lambda\}$. Let $0 < \varepsilon \leq 1/e^2$. Then, with probability at least $1 - \varepsilon$ over $x_1, \ldots, x_\lambda \sim X^\lambda$,*

$$\sum_{i=1}^{\lambda} \mathsf{H}_X[x_i] \geq \lambda\alpha - \mathrm{O}\left(\sqrt{\lambda \cdot \log\frac{1}{\varepsilon}} \cdot \log\left(|\mathcal{U}| \cdot \lambda\right)\right)$$

More specifically,

$$\sum_{i=1}^{\lambda} \mathsf{H}_X[x_i] \geq \lambda\alpha - \sqrt{8\lambda \cdot \ln\frac{1}{\varepsilon}} \cdot \left(\log|\mathcal{U}| + \frac{1}{2}\log\lambda\right)$$

When the random variables have linear entropy rate, i.e. $\mathsf{H}[X] = \Omega\left(|X|\right) = \Omega\left(\log|\mathcal{U}|\right)$, we prove the following corollary with explicit bound for λ.

Corollary 3.15. *If in Theorem 3.14 we let $\alpha = \kappa \cdot \log|\mathcal{U}| = \Omega\left(\log|\mathcal{U}|\right)$, then there exists $\lambda \leq \frac{32}{\kappa^2} \cdot \ln\frac{1}{\varepsilon}$ such that with probability at least $1 - \varepsilon$ over $x_1, \ldots, x_\lambda \sim X^\lambda$,*

$$\sum_{i=1}^{\lambda} \mathsf{H}_X[x_i] \geq \frac{1}{2}\lambda\alpha$$

Proof of Theorem 3.14. Standard concentration bounds do not apply on unbounded variables $\mathsf{H}_X(x_i)$'s. We lower bound $\sum_{i=1}^{\lambda} \mathsf{H}_X(x_i)$ in three steps: (i) introduce intermediate (bounded) variables $\mathsf{H}_X^{\geq\tau}(x_i) \stackrel{\text{def}}{=} \min\left\{\mathsf{H}_X(x_i), \log\frac{1}{\tau}\right\}$ for $\tau = \mathrm{O}\left(\frac{\log|\mathcal{U}|}{|\mathcal{U}|} \cdot \sqrt{\ln\frac{1}{\varepsilon}}\right)$

[3]The proof holds even when X_i's are independent but not identically distributed.

(to be specified later) to approximate $\mathsf{H}_X(x_i)$, (ii) upper bound the error of such an approximation, and (iii) apply Chernoff-Hoeffding Inequality on $\sum_{i=1}^{\lambda} \mathsf{H}_X^{\geq\tau}(x_i)$. Note that both steps (ii) and (iii) rely on different forms of measure concentration. Thus, the lower bound for $\sum_{i=1}^{\lambda} \mathsf{H}_X^{\geq\tau}(x_i)$ implies a lower bound for $\sum_{i=1}^{\lambda} \mathsf{H}_X(x_i)$.

From the definition of $\mathsf{H}_X^{\geq\tau}(x_i)$, we have $0 \leq \mathsf{H}_X^{\geq\tau}(x_i) \leq \log\frac{1}{\tau}$ and $\mathsf{H}_X^{\geq\tau}(x_i) \leq \mathsf{H}_X(x_i)$ with equality when $\mathsf{H}_X(x_i) \leq \log\frac{1}{\tau}$. Let $\Delta_i(x_i) \stackrel{\text{def}}{=} \mathsf{H}_X(x_i) - \mathsf{H}_X^{\geq\tau}(x_i) \geq 0$ and $\Delta_i \stackrel{\text{def}}{=} \Delta_i(X_i)$. Then, $\Delta_i(x_i) \geq q > 0$ holds if and only if $\Delta_i(x_i) = \mathsf{H}_X(x_i) - \mathsf{H}_X^{\geq\tau}(x_i) \geq q$ and $\mathsf{H}_X(x_i) > \mathsf{H}_X^{\geq\tau}(x_i) = \log\frac{1}{\tau}$, which is equivalent to $\mathsf{H}_X(x_i) \geq q + \log\frac{1}{\tau}$. As a result, for every $q > 0$ and for every $i \in \{1, \ldots, \lambda\}$,

$$\begin{aligned}
\Pr\left[\Delta_i \geq q\right] &= \Pr_{x_i \sim X_i}\left[\Delta_i(x_i) \geq q\right] \\
&= \Pr_{x_i \sim X_i}\left[\mathsf{H}_X(x_i) \geq q + \log\frac{1}{\tau}\right] = \sum_{\substack{x_i \in \mathcal{U}: \\ \mathsf{H}_X(x_i) \geq \log(2^q/\tau)}} \Pr[X_i = x_i] \\
&\leq |\mathcal{U}| \cdot 2^{-\log(2^q/\tau)} = \tau|\mathcal{U}| \cdot 2^{-q}
\end{aligned}$$

Let $\Delta \stackrel{\text{def}}{=} \sum_{i=1}^{\lambda} \Delta_i = \sum_{i=1}^{\lambda} \mathsf{H}_X(X_i) - \sum_{i=1}^{\lambda} \mathsf{H}_X^{\geq\tau}(X_i)$. We lower bound Δ using the Chernoff bound for random variables with exponential vanishing tails. To make the exposition self-contained, we provide the proof of Claim 3.16 (in our notation).

Claim 3.16 (Lemma A.2 in [Vad04])**.** *Let Δ, λ, τ as above. Let $\beta \stackrel{def}{=} \frac{e^{1/3}}{2^{1/2}} < 0.987$, then,*

$$\Pr\left[\Delta > 2\lambda\tau|\mathcal{U}|\right] \leq \beta^{\lambda\tau|\mathcal{U}|} = 2^{-\Omega(\lambda\tau|\mathcal{U}|)} \tag{3.5}$$

Proof of Claim 3.16. We consider the expectation of an exponential generating function as in usual Chernoff bounds. For every $t \in (0, 1)$ and for every $i \in \{1, \ldots, \lambda\}$,

$$\begin{aligned}
\mathrm{E}\left[2^{t\Delta_i}\right] &= \Pr\left[\Delta_i = 0\right] \cdot 2^0 + \int_{q=0}^{\infty} \frac{d}{dq}(-\tau|\mathcal{U}| \cdot 2^{-q}) \cdot 2^{tq} dq \\
&= 1 - \tau|\mathcal{U}| + \left(\frac{-\tau|\mathcal{U}|}{1-t} \cdot 2^{-(1-t)q}\right)\Bigg|_{q=0}^{\infty} \\
&= 1 + \frac{\tau|\mathcal{U}| \cdot t}{1-t} \leq e^{\frac{\tau|\mathcal{U}| \cdot t}{1-t}}
\end{aligned}$$

Thus, by the independence of the sampling of X_i's,

$$\mathrm{E}\left[2^{t\Delta}\right] = \prod_{i=1}^{\lambda} 2^{t\Delta_i} \leq \prod_{i=1}^{\lambda} \exp(\frac{\tau|\mathcal{U}| \cdot t}{1-t}) = e^{\frac{\lambda\tau|\mathcal{U}|\cdot t}{1-t}}$$

By Markov's Inequality, for every $t \in (0,1)$, we have

$$\Pr[\Delta > 2\lambda\tau|\mathcal{U}|] = \Pr[2^{t\Delta} > 2^{t\cdot 2\lambda\tau|\mathcal{U}|}] \leq \frac{e^{\frac{\lambda\tau|\mathcal{U}|\cdot t}{1-t}}}{2^{t\cdot 2\lambda\tau|\mathcal{U}|}} = \left(\frac{e^{t/(1-t)}}{2^{2t}}\right)^{\lambda\tau|\mathcal{U}|}$$

For $t = 1/4$, $e^{t/(1-t)}/2^{2t} = \beta < 1$, and the probability $\Pr[\Delta > 2\lambda\tau|\mathcal{U}|] \leq \beta^{\lambda\tau|\mathcal{U}|} = 2^{-\Omega(\lambda\tau|\mathcal{U}|)}$. □

To apply the concentration bound for $\sum_{i=1}^{\lambda} \mathsf{H}_X^{\geq\tau}(x_i)$, we need to first lower bound $\mathrm{E}\left[\sum_{i=1}^{\lambda} \mathsf{H}_X^{\geq\tau}(x_i)\right]$, where the expectation is over $(x_1, \ldots, x_\lambda) \sim X^\lambda$. Recall that for $x_1, \ldots, x_\lambda$ satisfying $\Delta \leq 2\lambda\tau|\mathcal{U}|$, there is $\sum_{i=1}^{\lambda} \mathsf{H}_X^{\geq\tau}(x_i) \geq \sum_{i=1}^{\lambda} \mathsf{H}_X(x_i) - 2\lambda\tau|\mathcal{U}|$. We denote by $p \stackrel{\text{def}}{=} \Pr[\Delta > 2\lambda\tau|\mathcal{U}|]$, and now we lower bound $\mathrm{E}\left[\sum_{i=1}^{\lambda} \mathsf{H}_X^{\geq\tau}(x_i)\right]$ in terms of $\sum_{i=1}^{\lambda} \mathsf{H}_X(x_i)$ as follows.

$$\begin{aligned}
\mathrm{E}\left[\sum_{i=1}^{\lambda} \mathsf{H}_X^{\geq\tau}(x_i)\right] &= (1-p)\cdot \mathrm{E}\left[\sum_{i=1}^{\lambda} \mathsf{H}_X^{\geq\tau}(x_i) \;\middle|\; \Delta \leq 2\lambda\tau|\mathcal{U}|\right] \\
&\quad + p \cdot \mathrm{E}\left[\sum_{i=1}^{\lambda} \mathsf{H}_X^{\geq\tau}(x_i) \;\middle|\; \Delta > 2\lambda\tau|\mathcal{U}|\right] \\
&\geq (1-p)\cdot \mathrm{E}\left[\sum_{i=1}^{\lambda} \mathsf{H}_X(x_i) - 2\lambda\tau|\mathcal{U}| \;\middle|\; \Delta \leq 2\lambda\tau|\mathcal{U}|\right] \\
&\geq (1-p)\cdot \mathrm{E}\left[\sum_{i=1}^{\lambda} \mathsf{H}_X(x_i) \;\middle|\; \Delta \leq 2\lambda\tau|\mathcal{U}|\right] - 2\lambda\tau|\mathcal{U}| \qquad (3.6)
\end{aligned}$$

Now, we lower bound $(1-p)\cdot \mathrm{E}\left[\sum_{i=1}^{\lambda} \mathsf{H}_X(x_i) \;\middle|\; \Delta \leq 2\lambda\tau|\mathcal{U}|\right]$. Since $\mathsf{H}_X(x_i) \geq \alpha$, we have $\mathrm{E}\left[\sum_{i=1}^{\lambda} \mathsf{H}_X(x_i)\right] = \sum_{i=1}^{\lambda} \mathsf{H}[X_i] \geq \lambda\alpha$, and by Jensen's Inequality[4] or by

[4]If X is a random variable and f is a convex function, then $f(\mathrm{E}[X]) \leq \mathrm{E}[f(X)]$.

multivariate calculus,

$$\begin{aligned}
&\sum_{\substack{\overline{x}=(x_1,\ldots,x_\lambda)\in\mathcal{U}^\lambda \\ \sum_{i=1}^{\lambda}\Delta_i(x_i)>2\lambda\tau|\mathcal{U}|}} \Pr[X^\lambda=\overline{x}]\cdot\log\frac{1}{\Pr[X^\lambda=\overline{x}]} \\
\leq&|\mathcal{U}|^\lambda\cdot\frac{\Pr\left[\Delta>2\lambda\tau|\mathcal{U}|\right]}{|\mathcal{U}|^\lambda}\cdot\log\frac{|\mathcal{U}|^\lambda}{\Pr\left[\Delta>2\lambda\tau|\mathcal{U}|\right]} \\
=&p\cdot\log\frac{|\mathcal{U}|^\lambda}{p} \\
=&p\cdot\left(\lambda\log|\mathcal{U}|+\log\frac{1}{p}\right)
\end{aligned}$$

Therefore,

$$\begin{aligned}
\lambda\alpha &\leq \mathrm{E}\left[\sum_{i=1}^{\lambda}\mathsf{H}_X(x_i)\right] \\
&= (1-p)\cdot\mathrm{E}\left[\sum_{i=1}^{\lambda}\mathsf{H}_X(x_i)\;\middle|\;\Delta\leq 2\lambda\tau|\mathcal{U}|\right] \\
&\quad + \quad p\cdot\mathrm{E}\left[\sum_{i=1}^{\lambda}\mathsf{H}_X(x_i)\;\middle|\;\Delta>2\lambda\tau|\mathcal{U}|\right] \\
&\leq (1-p)\cdot\mathrm{E}\left[\sum_{i=1}^{\lambda}\mathsf{H}_X(x_i)\;\middle|\;\Delta\leq 2\lambda\tau|\mathcal{U}|\right] \\
&\quad + \sum_{\substack{\overline{x}=(x_1,\ldots,x_\lambda)\in\mathcal{U}^\lambda \\ \sum_{i=1}^{\lambda}\Delta_i(x_i)>2\lambda\tau|\mathcal{U}|}} \Pr[X^\lambda=\overline{x}]\cdot\log\frac{1}{\Pr[X^\lambda=\overline{x}]} \\
&\leq (1-p)\cdot\mathrm{E}\left[\sum_{i=1}^{\lambda}\mathsf{H}_X(x_i)\;\middle|\;\Delta\leq 2\lambda\tau|\mathcal{U}|\right]+p\cdot\left(\lambda\log|\mathcal{U}|+\log\frac{1}{p}\right) \qquad (3.7)
\end{aligned}$$

Combining (3.6) and (3.7), we get the following lower bound:

$$\mathrm{E}\left[\sum_{i=1}^{\lambda}\mathsf{H}_X^{\geq\tau}(x_i)\right]\geq\lambda\alpha-p\cdot\left(\lambda\log|\mathcal{U}|+\log\frac{1}{p}\right)-2\lambda\tau|\mathcal{U}| \qquad (3.8)$$

Now, we apply a standard form of Chernoff-Hoeffding Inequality on $\sum_{i=1}^{\lambda}\mathsf{H}_X^{\geq\tau}(x_i)$,

where $0 \le \mathsf{H}_X^{\ge\tau}[x_i] \le \log\frac{1}{\tau}$ for every $x_i \in \mathcal{U}$ and $i \in \{1,\dots,\lambda\}$.

$$\begin{aligned}
&\Pr\left[\sum_{i=1}^{\lambda}\mathsf{H}_X^{\ge\tau}(x_i) \le \mathrm{E}\left[\sum_{i=1}^{\lambda}\mathsf{H}_X^{\ge\tau}(x_i)\right] - \delta\right] \\
&= \Pr\left[-\sum_{i=1}^{\lambda}\mathsf{H}_X^{\ge\tau}(x_i) + \mathrm{E}\left[\sum_{i=1}^{\lambda}\mathsf{H}_X^{\ge\tau}(x_i)\right] \ge \delta\right] \\
&\le \exp\left(-\frac{2\delta^2}{\sum_{i=1}^{\lambda}(\log\frac{1}{\tau}-0)^2}\right) \\
&= \exp\left(-\frac{2\delta^2}{\lambda(\log\frac{1}{\tau})^2}\right)
\end{aligned}$$

For $\delta = \log\frac{1}{\tau}\sqrt{\frac{\lambda}{2}\ln\frac{1}{\varepsilon}}$, we have

$$\begin{aligned}
&\Pr\left[\sum_{i=1}^{\lambda}\mathsf{H}_X^{\ge\tau}(x_i) \le \mathrm{E}\left[\sum_{i=1}^{\lambda}\mathsf{H}_X^{\ge\tau}(x_i)\right] - \delta\right] \\
&= \Pr\left[\sum_{i=1}^{\lambda}\mathsf{H}_X^{\ge\tau}(x_i) \le \mathrm{E}\left[\sum_{i=1}^{\lambda}\mathsf{H}_X^{\ge\tau}(x_i)\right] - \log\frac{1}{\tau}\sqrt{\frac{\lambda}{2}\ln\frac{1}{\varepsilon}}\right] \\
&\le \exp\left(-\frac{2\delta^2}{\lambda(\log\frac{1}{\tau})^2}\right) = \varepsilon
\end{aligned}$$

Recalling that $\sum_{i=1}^{\lambda}\mathsf{H}_X(x_i) \ge \sum_{i=1}^{\lambda}\mathsf{H}_X^{\ge\tau}(x_i)$ and the lower bound in (3.8), the following inequality holds with probability at least $1-\varepsilon$ over $x_1,\dots,x_\lambda \sim X^\lambda$,

$$\begin{aligned}
\sum_{i=1}^{\lambda}\mathsf{H}_X(x_i) &\ge \sum_{i=1}^{\lambda}\mathsf{H}_X^{\ge\tau}(x_i) \\
&\ge \lambda\alpha - 2\lambda\tau|\mathcal{U}| - p\cdot\left(\lambda\log|\mathcal{U}| + \log\frac{1}{p}\right) - \delta
\end{aligned} \tag{3.9}$$

where δ and τ depends on ε.

Finally, we fix parameters and conclude the proof. For the given λ and ε, select τ such that

$$\lambda = \frac{\log^2\frac{1}{\tau}}{2\tau^2|\mathcal{U}|^2}\cdot\ln\frac{1}{\varepsilon}$$

Since $\lambda \in \mathbb{Z}^+$, we have $\tau = \mathrm{O}\left(\frac{\log|\mathcal{U}|}{|\mathcal{U}|} \cdot \sqrt{\ln\frac{1}{\varepsilon}}\right)$. We let $c \stackrel{\text{def}}{=} \tau|\mathcal{U}|/\log|\mathcal{U}| = \mathrm{O}\left(\sqrt{\ln\frac{1}{\varepsilon}}\right)$, and thus,

$$\tau = \frac{c\log|\mathcal{U}|}{|\mathcal{U}|}$$

Then, $\delta = \lambda\tau|\mathcal{U}|$. Moreover, $\lambda\tau|\mathcal{U}| \geq p \cdot \left(\lambda\log|\mathcal{U}| + \log\frac{1}{p}\right)$, since

$$\lambda = \frac{\log^2\frac{1}{\tau}}{2\tau^2|\mathcal{U}|^2}\ln\frac{1}{\varepsilon} = \frac{\left(\log\frac{|\mathcal{U}|}{c\log|\mathcal{U}|}\right)^2}{2(c\log|\mathcal{U}|)^2}\ln\frac{1}{\varepsilon} = \Omega\left(\left(\frac{1}{c}\right)^2\ln\frac{1}{\varepsilon}\right) \geq \frac{\log\left(\frac{1}{c}+\log\frac{1}{\beta}\right)}{c\log|\mathcal{U}|\cdot\log\frac{1}{\beta}}$$

$$\Longrightarrow \lambda\tau|\mathcal{U}| \geq \frac{\log\left(\frac{\log|\mathcal{U}|}{\tau|\mathcal{U}|}+\log\frac{1}{\beta}\right)}{\log\frac{1}{\beta}}$$

$$\Longrightarrow 1 \geq \beta^{\lambda\tau|\mathcal{U}|}\cdot\left(\frac{\log|\mathcal{U}|}{\tau|\mathcal{U}|}+\log\frac{1}{\beta}\right)$$

$$\Longrightarrow \lambda\tau|\mathcal{U}| \geq \beta^{\lambda\tau|\mathcal{U}|}\cdot\left(\lambda\log|\mathcal{U}|+\log\frac{1}{\beta^{\lambda\tau|\mathcal{U}|}}\right) \geq p\cdot\left(\lambda\log|\mathcal{U}|+\log\frac{1}{p}\right)$$

where the last inequality holds because $p \leq \beta^{\lambda\tau|\mathcal{U}|}$ by (3.5).

Thus, by (3.9) we have

$$\Pr_{x_1,\ldots,x_i\sim X^\lambda}\left[\sum_{i=1}^{\lambda}\mathsf{H}_X(x_i) \geq \lambda\alpha - 4\lambda\tau|\mathcal{U}|\right] \geq 1-\varepsilon \tag{3.10}$$

For the general case, we upper bound $\lambda\tau|\mathcal{U}|$ by $\mathrm{O}\left(\sqrt{\lambda\cdot\log(1/\varepsilon)}\cdot\log\left(|\mathcal{U}|\cdot\lambda\right)\right)$ as follows. Recall that $c \leq \mathrm{O}\left(\log|\mathcal{U}|\right)$ and hence $\tau < 1, \log\log\frac{1}{\tau} > 0$, therefore for

$\varepsilon \leq 1/e^2$,

$$\begin{aligned}
&\log\log\frac{1}{\tau} + \frac{1}{2}\left(\log\ln\frac{1}{\varepsilon} - \log 2\right) \geq 0 \\
\implies &\log\log\frac{1}{\tau} + \frac{1}{2}\left(\log\ln\frac{1}{\varepsilon} - \log 2\right) + \log|\mathcal{U}| - \log c - \log\log|\mathcal{U}| \\
&\geq \log|\mathcal{U}| - \log c - \log\log|\mathcal{U}| \\
\implies &\log|\mathcal{U}| + \frac{1}{2}\log\left(\frac{(\log\frac{1}{\tau})^2}{2(c\log|\mathcal{U}|)^2}\ln\frac{1}{\varepsilon}\right) \geq \log\left(\frac{|\mathcal{U}|}{c\log|\mathcal{U}|}\right) \\
\implies &\log|\mathcal{U}| + \frac{1}{2}\log\lambda \geq \log\frac{1}{\tau} \\
\implies &\frac{\log|\mathcal{U}| + \frac{1}{2}\log\lambda}{\log\frac{1}{\tau}} \cdot \lambda\tau|\mathcal{U}| \geq \lambda\tau|\mathcal{U}|
\end{aligned}$$

Notice that $\frac{\lambda\tau|\mathcal{U}|}{\log\frac{1}{\tau}} = \sqrt{\frac{1}{2}\lambda\ln\frac{1}{\varepsilon}}$ in the above inequality, and thus we conclude

$$\lambda\tau|\mathcal{U}| \leq \left(\log|\mathcal{U}| + \frac{1}{2}\log\lambda\right)\cdot\sqrt{\frac{1}{2}\lambda\ln\frac{1}{\varepsilon}} = \mathrm{O}\left(\sqrt{\lambda\cdot\log\frac{1}{\varepsilon}}\cdot\log\left(|\mathcal{U}|\cdot\lambda\right)\right)$$

By plugging in this upper bound for $\lambda\tau|\mathcal{U}|$ into (3.10), we have that with probability at least $1-\varepsilon$ over $x_1,\ldots,x_\lambda \sim X^\lambda$, the following holds:

$$\begin{aligned}
\sum_{i=1}^{\lambda}\mathsf{H}_X(x_i) \geq &\lambda\alpha - 4\lambda\tau|\mathcal{U}| \\
\geq &\lambda\alpha - \sqrt{8\lambda\cdot\ln\frac{1}{\varepsilon}}\cdot\left(\log|\mathcal{U}| + \frac{1}{2}\log\lambda\right) \\
= &\lambda\alpha - \mathrm{O}\left(\sqrt{\lambda\cdot\log\frac{1}{\varepsilon}}\cdot\log\left(|\mathcal{U}|\cdot\lambda\right)\right)
\end{aligned}$$

This completes the proof for Theorem 3.14. □

Then, we prove Corollary 3.15 with the explicit constants.

Proof of Corollary 3.15. By Theorem 3.14, for $\alpha = \Omega\left(\log|\mathcal{U}|\right)$ and let $c = \frac{\alpha}{8\log|\mathcal{U}|}$ be

a constant, with probability at least $1-\varepsilon$, we have:

$$\sum_{i=1}^{\lambda} \mathsf{H}_X[x_i] \geq \lambda\alpha - 4\lambda\tau|\mathcal{U}|$$

Recalling that in the proof of Theorem 3.14,

$$\tau = \frac{c\log|\mathcal{U}|}{|\mathcal{U}|} = \frac{\alpha}{8|\mathcal{U}|}$$
$$\lambda = \frac{\log^2\frac{1}{\tau}}{2\tau^2|\mathcal{U}|^2}\cdot\ln\frac{1}{\varepsilon}$$

We lower bound $\sum_{i=1}^{\lambda}\mathsf{H}_X[x_i]$ as follows,

$$\sum_{i=1}^{\lambda} \mathsf{H}_X[x_i] \geq \lambda\alpha - 4\lambda\tau|\mathcal{U}| = \lambda\alpha - 4\lambda\cdot\frac{\alpha}{8} = \frac{1}{2}\lambda\alpha$$

On the other hand, λ is upper bounded as follows,

$$\begin{aligned}\lambda &= \frac{\log^2\frac{1}{\tau}}{2\tau^2|\mathcal{U}|^2}\cdot\ln\frac{1}{\varepsilon} = \frac{\log^2(8|\mathcal{U}|/\alpha)}{2(\alpha/8)^2}\cdot\ln\frac{1}{\varepsilon}\\ &= \frac{\left(3+\log|\mathcal{U}|-\log\alpha\right)^2}{\alpha^2/32}\cdot\ln\frac{1}{\varepsilon}\\ &\leq 32\cdot\left(\frac{\log|\mathcal{U}|}{\alpha}\right)^2\cdot\ln\frac{1}{\varepsilon} = \frac{32}{\kappa^2}\cdot\ln\frac{1}{\varepsilon}\end{aligned}$$

□

3.2.4 Streaming Extraction from Any Source

We show the last and most general property of RRB, which holds for arbitrary sources. In particular, we show that RRB extracts randomness when given (on the same stream one after the other) $\mathrm{O}\left(\log\frac{n}{\varepsilon}\right)$ many samples from independent arbitrary (n,k)-sources. Actually, we believe that RRB works even on a single sample, as concluded in Conjecture 4.13 on page 115.

Theorem 3.17. *For every $\varepsilon > 0$ and $k = \Omega(n)$, there is a number of sources $\lambda =$*

$\mathrm{O}\left(\log \frac{n}{\varepsilon}\right)$, *such that for* $b = \Theta(\lambda \log \frac{n}{\varepsilon}), d = \mathrm{O}\left(b \log n\right), \ell = \Omega\left(n/\log^2 \frac{n}{\varepsilon}\right)$, *and* $m = \Omega\left(\lambda n/\log \frac{n}{\varepsilon}\right)$, *we have that* $\mathsf{RRB} : \{0,1\}^{\lambda \times n} \times \{0,1\}^d \to \{0,1\}^m$ *is a* (k,ε)*-extractor for* λ *many independent* (n,k)*-sources. Here, the samples of these sources are listed one after the other in the same stream, which is the input stream for* RRB.

Proof. Let the input to RRB be partitioned as $X = (X^{(1)}, X^{(2)}, \ldots, X^{(\lambda)})$ where $X^{(1)}, \ldots, X^{(\lambda)}$ are independent random variables over $\{0,1\}^n$ and $\mathsf{H}_\infty[X^{(i)}] \geq k$ for every $i \in [\lambda]$. We want to show that $\mathcal{SD}\left(\mathsf{RRB}\left(X^{(1)}, \ldots, X^{(\lambda)}\right), Y\right), \mathcal{U}_m\right) \leq \varepsilon$ when $Y \sim \mathcal{U}_d$. For convenience, let $b' = b/\lambda$ and partition $Y = (Y_0, Y_1, \ldots, Y_b) = \left(Y_0, Y^{(1)}, \ldots, Y^{(\lambda)}\right)$ where $Y^{(i)} = \left(Y_1^{(i)}, \ldots, Y_{b'}^{(i)}\right) \stackrel{\text{def}}{=} \left(Y_{(i-1)b'+1}, \ldots, Y_{ib'}\right)$ is the randomness used to shift blocks in $X^{(i)}$ for every $i \in [\lambda]$. And similarly to the partition of Y, we also partition $X = \left(\overline{X}_1, \ldots, \overline{X}_b\right)$ where $X^{(i)} = \left(\overline{X}_{(i-1)b'+1}, \ldots, \overline{X}_{ib'}\right)$.

Input. In this construction RRB runs as if X is a single source (see description of RRB on page 59), where the parameters and the length of $X^{(1)}, \ldots, X^{(\lambda)}$ are set in a way such that each $X^{(i)}$ corresponds to a super-block in the RRB description.

The proof consists of three parts:

i. after Stage I and II in RRB, we get an *interlace* (defined below) of $Z^{(1)}, \ldots, Z^{(\lambda)}$;

ii. with many independent samples the next-block-entropy guarantee of RRB (as in Lemma 3.10, p. 65) transforms to next-block-min-entropy guarantee;

iii. Stage III applies a hash function to extract randomness from guaranteed next-block-min-entropy (Lemma 3.9, p. 63).

Definition 3.18. *Suppose* $Z^{(1)}, \ldots, Z^{(\lambda)}$ *are partitioned into blocks such that for every* $i \in [\lambda]$, $Z^{(i)} = \left(Z_1^{(i)}, \ldots, Z_\ell^{(i)}\right)$. *An* interlace *of* $Z^{(1)}, \ldots, Z^{(\lambda)}$ *makes up its* i*-th block by collecting the* i*-th block from each of the* $Z^{(1)}, \ldots, Z^{(\lambda)}$ *as follows.*

$$\mathsf{interlace}(Z^{(1)}, \ldots, Z^{(\lambda)}) = \left((Z_1^{(1)}, \ldots, Z_1^{(\lambda)}), \ldots, (Z_\ell^{(1)}, \ldots, Z_\ell^{(\lambda)})\right)$$

To prove (1) above we identify the execution of Stage I and II in RRB by a different algorithm that results in an identical execution to that of RRB. Consider the run of

RRB until the end of Step 5 on input $X^{(i)} = \left(\overline{X}_1^{(i)}, \ldots, \overline{X}_{b'}^{(i)}\right)$ for $b' = b/\lambda$ and get $Z^{(i)} = \left(Z_1^{(i)}, \ldots, Z_{n/b'}^{(i)}\right)$, for every $i \in [\lambda]$. Note here $\overline{X}_j^{(i)} = \overline{X}_{(i-1)b'+j}$ for every $i \in [\lambda]$ and $j \in [b']$ since $X^{(i)} = \left(\overline{X}_{(i-1)b'+1}, \ldots, \overline{X}_{ib'}\right)$.

Then, realize every $Z^{(1)}, \ldots, Z^{(\lambda)}$ as super-blocks and do re-bucketing on $Z_j^{(i)}$'s for $i \in [\lambda]$ and $j \in [n/b']$. Thus, we obtain $\left((Z_1^{(1)}, \ldots, Z_1^{(\lambda)}), \ldots, (Z_{n/b'}^{(1)}, \ldots, Z_{n/b'}^{(\lambda)})\right)$, which is exactly the interlace of $Z^{(1)}, \ldots, Z^{(\lambda)}$.

Recalling that $Z_j^{(i)} = \left(\mathsf{shift}(\overline{X}_1^{(i)}, Y_1^{(i)})_j, \ldots, \mathsf{shift}(\overline{X}_{b'}^{(i)}, Y_{b'}^{(i)})_j\right)$ and hence

$$\begin{aligned}
\left(Z_j^{(1)}, \ldots, Z_j^{(\lambda)}\right) =& \Big(\left(\mathsf{shift}(\overline{X}_1^{(1)}, Y_1^{(1)})_j, \ldots, \mathsf{shift}(\overline{X}_{b'}^{(1)}, Y_{b'}^{(1)})_j\right), \\
& \qquad \ldots, \left(\mathsf{shift}(\overline{X}_1^{(\lambda)}, Y_1^{(\lambda)})_j, \ldots, \mathsf{shift}(\overline{X}_{b'}^{(\lambda)}, Y_{b'}^{(\lambda)})_j\right)\Big) \\
=& \Big(\left(\mathsf{shift}(\overline{X}_1, Y_1)_j, \ldots, \mathsf{shift}(\overline{X}_{b'}, Y_{b'})_j\right), \\
& \qquad \ldots, \left(\mathsf{shift}(\overline{X}_{(\lambda-1)b'+1}, Y_{(\lambda-1)b'+1}, \ldots, \mathsf{shift}(\overline{X}_{\lambda b'}, Y_{\lambda b'})_j\right)\Big) \\
=& \left(\mathsf{shift}(\overline{X}_1, Y_1)_j, \ldots, \mathsf{shift}(\overline{X}_{\lambda b'}, Y_{\lambda b'})_j\right) \\
=& \left(\mathsf{shift}(\overline{X}_1, Y_1)_j, \ldots, \mathsf{shift}(\overline{X}_b, Y_b)_j\right)
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\mathsf{interlace}\left(Z^{(1)}, \ldots, Z^{(\lambda)}\right) = \left(\left(Z_1^{(1)}, \ldots, Z_1^{(\lambda)}\right), \ldots, \left(Z_{n/b'}^{(1)}, \ldots, Z_{n/b'}^{(\lambda)}\right)\right) \\
=& \left(\mathsf{shift}(\overline{X}_1, Y_1)_1, \ldots, \mathsf{shift}(\overline{X}_b, Y_b)_1, \ldots, \mathsf{shift}(\overline{X}_1, Y_1)_{n/b'}, \ldots, \mathsf{shift}(\overline{X}_b, Y_b)_{n/b'}\right) \\
=& \mathsf{interlace}\left(\mathsf{shift}(\overline{X}_1, Y_1), \ldots, \mathsf{shift}(\overline{X}_b, Y_b)\right)
\end{aligned}$$

which is exactly the intermediate result in Step 5 of RRB on input (X, Y) (partitioned into b super-blocks). Therefore, with the same algorithm (RRB) we can get the interlace of $Z^{(1)}, \ldots, Z^{(\lambda)}$ derived from λ many independent random variables (each partitioned into $b' = b/\lambda$ super-blocks).

In the remainder of the proof, our analysis is about $Z = \left(Z_1, \ldots, Z_{n/b'}\right) = \left((Z_1^{(1)}, \ldots, Z_1^{(\lambda)}), \ldots, (Z_{n/b'}^{(1)}, \ldots, Z_{n/b'}^{(\lambda)})\right)$ as computed with this new sub-algorithm (which replaces Stage I and II), and by equivalence the same analysis applies to the execution of RRB.

Now, let us show how the next-block-entropy guarantee is converted to next-block-min-entropy guarantee. By Lemma 3.10 there exists $b' = \Theta(\log \frac{n}{\varepsilon})$ and $\ell = \Omega\left(\frac{n}{\log^2 \frac{n}{\varepsilon}}\right)$ such that $\left(Z_1^{(i)}, \ldots, Z_\ell^{(i)}\right)$ has next-block-entropy $\alpha = \Omega\left(\log \frac{n}{\varepsilon}\right)$ with probability at least $1 - \frac{\varepsilon}{n}$ over the random choice of $Y^{(i)}$. As a result the above statement holds for every $i \in [\lambda]$ with probability at least $1 - \frac{\lambda\varepsilon}{n}$. That is, by suffering a loss of $\frac{\lambda\varepsilon}{n}$, the argument goes through conditioned on the premise that all $Z^{(i)}$'s have next-block-entropy guarantee α.

Notice that independent random variables with next-block-entropy α implies next-block-min-entropy by Corollary 3.15 (page 71). Fix $j \in [\ell]$ and apply this corollary on $Z^{(1)}, \ldots, Z^{(\lambda)}$. Then, since $\mathsf{H}\left[Z_j^{(i)} \middle| Z_{j-1}^{(i)}, \ldots, Z_1^{(i)}\right] \geq \alpha$ for every $i \in [\lambda]$, there must be $\tilde{Z}_j$ such that $\mathcal{SD}\left(\left(\tilde{Z}_j, Z_{j-1}, \ldots, Z_1\right), \left(Z_j, Z_{j-1}, \ldots, Z_1\right)\right) \leq \varepsilon' + 2^{-\Omega(\lambda)}$ and

$$\mathsf{H}_\infty\left[\tilde{Z}_j \middle| Z_{j-1}, \ldots, Z_1\right] \geq \frac{1}{2}\lambda\alpha$$

Therefore, when $\lambda = \Omega\left(\log \frac{n}{\varepsilon}\right), \alpha = \Omega\left(\log \frac{n}{\varepsilon}\right), b' = \Theta(\log \frac{n}{\varepsilon})$ and $\varepsilon' = \frac{\varepsilon}{n}$, there exists $\beta = \Omega(\lambda\alpha)$ and $\tilde{Z}_1, \ldots, \tilde{Z}_\ell$ such that $\mathcal{SD}\left(\left(\tilde{Z}_1, \ldots, \tilde{Z}_\ell\right), \left(Z_1, \ldots, Z_\ell\right)\right) \leq \frac{\varepsilon\ell}{n} + \frac{\lambda\varepsilon}{n}$ with next-block-min-entropy guarantee

$$\mathsf{H}_\infty\left[\tilde{Z}_j \middle| \tilde{Z}_{j-1}, \ldots, \tilde{Z}_1\right] \geq \beta$$

Finally, setting $\rho = 2\log \frac{n}{\varepsilon}$ in Lemma 3.9 we conclude that

$$\begin{aligned}
&\mathcal{SD}\left(\mathsf{RRB}\left(X, Y\right), \mathcal{U}_m\right) = \mathcal{SD}\left(\left(H(Z_1), \ldots, H(Z_\ell)\right), \mathcal{U}_m\right) \\
\leq &\mathcal{SD}\left(\left(H(\tilde{Z}_1), \ldots, H(\tilde{Z}_\ell)\right), \mathcal{U}_m\right) + \frac{\varepsilon}{2} + \frac{\lambda\varepsilon}{n} \\
\leq &2^{-\rho/2} \cdot \ell + \frac{\varepsilon}{2} + \frac{\lambda\varepsilon}{n} = \frac{\varepsilon\ell}{n} + \frac{\varepsilon}{2} + \frac{\lambda\varepsilon}{n} \leq \varepsilon
\end{aligned}$$

where H is uniformly chosen (with random seed Y_0) from a family of 2-universal hash functions mapping b bits to $\beta - \rho$ bits. Recalling that $\ell = \Omega\left(n/\log^2 \frac{n}{\varepsilon}\right)$, $\alpha = \Omega\left(\log \frac{n}{\varepsilon}\right)$ and $\beta = \Omega(\lambda\alpha)$, the output length $m = (\beta - \rho)\ell = \Omega(\lambda\alpha\ell) = \Omega\left(\frac{\lambda n}{\log \frac{n}{\varepsilon}}\right)$. Therefore, $\mathsf{RRB} : \{0,1\}^{\lambda n} \times \{0,1\}^d \to \{0,1\}^m$ is the extractor from λ many independent (n, k)

sources as stated in Theorem 3.17. □

3.3 Experimental Study

The complete empirical method consists of: (i) initial randomness generation, (ii) parameter estimation, and (iii) streaming extraction. Components (ii) and (iii) rely on initial randomness.

Initial randomness is generated in two phases. First, extract randomness from multiple independent sources without using any seed. Then, use RRB to expand this initial randomness further.

Parameter estimation determines a suitable pair (κ, γ) of min-entropy rate $\kappa \stackrel{\text{def}}{=} k/n$ and effectiveness factor $\gamma \stackrel{\text{def}}{=} b_O/(\frac{n}{b})$.

The heart of our method is the RRB extractor realized as a two-stream algorithm with each stream maintained on a different hard disk. RRB has provable guarantees and is highly practical.

3.3.1 Experimental set-up.

We empirically evaluate the quality and the efficiency of our method.

Quality evaluation is performed on big samples from 12 semantic data categories: compressed/uncompressed audio, video, images, text, DNA sequenced data, and social network data (for audio, video, and images the compression is lossy). The initial randomness used in our experiments consists of 9.375×10^8 bits $\approx$ 117 MB generated from 144 pieces of 4 MB compressed audio and one piece of 15 GB compressed video. The produced randomness is used for parameter estimation on samples ranging in size from 1 GB to 16 GB from each of the 12 categories. The estimated κ and γ vary within $[1/64, 1/2]$ and $[1/32, 1/2]$ respectively, cross-validated (i.e. excluding previously used samples) on samples of size 1.5 GB – 20 GB with error tolerance $\varepsilon = 10^{-20}$. Final extraction quality is measured on all 12 categories by NIST and DIEHARD.

Operating system kernel-level measurements are taken for the running time and

memory usage of RRB. These measurements are taken from RRB on input sizes 1 GB – 20 GB, min-entropy rate $\kappa \in \{1/4, 1/8\}$, and error tolerance $\varepsilon \in \{10^{-10}, 10^{-20}\}$.

For comparison, we measure quality and efficiency for three of the most popular representatives of extractors. The quality of Local Hash and von Neumann extractor is evaluated on 12 GB raw data (from the 12 categories) and 12 GB adversarial synthetic data (Table 3.2). The efficiency is measured for von Neumann extractor, Local Hash, and Trevisan's extractor (Figure 3-2).

3.3.2 Empirical initial randomness generation.

Seeded extraction, as in RRB, needs uniform random bits to start. All the randomness for the seeds in our experiments is obtained by *randomness bootstrapping* in two phases: (i) obtain initial randomness ρ through (seedless) multiple-independent-source extraction, and (ii) use ρ for parameter estimation and run RRB to extract a longer string ρ_{long}, $|\rho_{\mathsf{long}}| = 2^{\sqrt{|\hat{\rho}|/54}-7}$, where $\hat{\rho}$ is the part of ρ used as the seed of RRB during bootstrapping. By elementary information theory, ρ_{long} can be used instead of a uniformly random string.

Phase (i) is not known to have a streaming implementation, which is a not an issue since it only extracts from small samples. Start with 144 statistically independent compressed audio samples $\rho_1, \ldots, \rho_{144}$: each sample is 4 MB of high-quality (320 Kbps) compressed recording (MPEG2-layer3). Taken together, the samples last 4.1 hours. These samples are generated privately – without malicious adversarial control – using different independent sound-settings and sources. Partition the samples into 16 groups, each consisting of $9 = 3^2$ samples. Every ρ_i can be interpreted as a field element in $\mathsf{GF}[p]$, where $p = 2^{57885161}-1$ is the largest known Mersenne prime and 4 MB $< \log_2 p$ bits. For the first group $\{\rho_1, \ldots, \rho_9\}$, compute $\rho^{(1)} = \rho' \cdot \rho'' + \rho'''$ where $\rho' = \rho_1 \cdot \rho_2 + \rho_3$, $\rho'' = \rho_4 \cdot \rho_5 + \rho_6$, and $\rho''' = \rho_7 \cdot \rho_8 + \rho_9$; which is a two-level recursion. In the same way, compute $\rho^{(2)}, \ldots, \rho^{(16)}$ and finally let $\rho = \rho^{(1)} + \cdots + \rho^{(16)}$, with all operations in $\mathsf{GF}[p]$. We call this the BIWZ method due to the authors [BIW06, Zuc90] who studied provable multi-source extraction based on the field operation $x \cdot y + z$.

Phase (ii) uses the 4 MB extracted by BIWZ out of which 3.99 MB are used

in parameter estimation for compressed video. The remaining 10 KB are used to run RRB on 15 GB compressed video, which is generated and compressed privately, i.e. without malicious adversarial control. Our hypothesis is that the estimated parameters are valid for RRB, i.e. n bits of compressed video contain min-entropy $n/2$ that can be extracted by RRB with effectiveness factor $\gamma = 1/32$. This hypothesis is verified experimentally (Table 3.1). With the given seed and $\kappa = 1/2, \gamma = 1/32$, and $\varepsilon = 10^{-100}$, RRB extracts the final 9.375×10^8 random bits.

Here is why BIWZ works. Each $\rho^{(i)}$ has provable extraction guarantees [BIW06] for sources of relatively high min-entropy (e.g. $\geq 0.9 \log_2 p$), and the same method works for any linear min-entropy; e.g. in theory, 10^9 levels suffice for min-entropy $n/8$. Each $\rho^{(i)}$ alone is empirically measured to be indistinguishable from *ideal uniform random bits*. By summing up the $\rho^{(i)}$'s, the statistical distance provably reduces to the minimum among all sources associated with the $\rho^{(i)}$'s. In the summation for ρ, 16 is a safety factor. BIWZ cannot work on samples > 7 MB because the required prime p exceeds current human knowledge. Note that mathematically deep followup works are quite limited in practice even on sample size $n = 8 \times 10^6$ bits= 1 MB because of e.g. arithmetic over GF $[2^n]$.

3.3.3 Empirical parameter estimation protocol.

There are two crucial parameters for RRB: the min-entropy rate κ and the effectiveness factor γ. In theory, γ is determined by κ, n, ε. In practice, better, empirically validated values are estimated simultaneously for κ and γ. This works because in addition to min-entropy, κ induces the next-block-min-entropy guarantee for a fraction of γ blocks (Figure 3-1).

For every semantic data category, the following protocol estimates a pair of (κ, γ).

First, get a bit sequence s of size 1 GB by concatenating sampled < 1 MB segments from the target data category. Then, compress s into s' using LZ77 [ZL77] ($s' = s$ if s is already compressed). Since the ideal compression has $|s'|$ equal to the Shannon entropy of s, the compression rate $\frac{|s'|}{|s|}$ is also an upper bound for the min-entropy rate. To obtain a lower bound for the min-entropy rate (required parameter for

RRB), we start from $\kappa' = \frac{|s'|}{2|s|}$ and search inside $[0, \kappa']$. For min-entropy rate $\kappa \in \left\{\kappa', \frac{\kappa'}{2}, \frac{\kappa'}{4}, \frac{\kappa'}{8}, \frac{\kappa'}{16}\right\}$ and effectiveness factor $\gamma \in \left\{\gamma', \frac{\gamma'}{2}, \frac{\gamma'}{4}, \frac{\gamma'}{8}, \frac{\gamma'}{16}\right\}$, extract from s using RRB, with parameters κ, γ, and $\varepsilon = 10^{-20}$ and seed from the initial randomness. Apply NIST tests on the extracted bits for every (κ, γ) pair. If the amount of extracted bits is insufficient for NIST tests, then start over with an s twice as long. We call a pair of (κ_0, γ_0) *acceptable* if NIST fails with frequency at most 0.25% for every run of RRB with parameters $\kappa \leq \kappa_0$ and $\gamma \leq \gamma_0$. This 0.25% threshold is conservatively set slightly below the expected failure probability of NIST on ideal random inputs, which is 0.27%. If (κ_0, γ_0) is a correctly estimated lower bound, then every estimate (κ, γ) with $\kappa \leq \kappa_0$ and $\gamma \leq \gamma_0$ is also a correct lower bound. Hence, the extraction with (κ, γ) should be random and pass the NIST tests. We choose the acceptable pair (if any) that maximizes the output length.

There is strong intuition in support of the correct operation of this protocol. First, the random sampling for s preserves with high probability the min-entropy rate [NZ96]. Second, an extractor cannot extract almost-uniform randomness if the source has min-entropy much lower than the estimated one. Finally, NIST tests exhibit a certain ability to detect non-uniformity. Verification of the estimated parameters is done by cross-validation.

3.3.4 Streaming realization of the RRB extractor.

The streaming extractor RRB, as defined in Section 3.1 (page 57), uses $d + 2\log_2 n$ bits local memory and $3\lceil \log_2 \log_2 \frac{n}{\varepsilon} - 2\log_2 \kappa + 2\log_2 3 + 1 \rceil$ passes over two streams, for input length n, min-entropy rate κ, error tolerance ε and seed length d. RRB is also parameterized by the effectiveness factor γ as shown below.

Given n, ε, and the estimated κ, γ, our method invokes RRB with $k = \kappa n$, $m = \frac{1}{2}\gamma\kappa n$, and the number of super-blocks $b = \frac{9}{2\kappa^2}\log_2 \frac{n}{\varepsilon}$. For convenience, n is padded to a power of 2, κ and γ are rounded down to an inverse power of 2, and b is rounded up to a power of 2. Hereafter, no further rounding is needed. Let σ_1 and σ_2 denote two read/write streams. The input sample $x \in \{0,1\}^n$ is initially on σ_1. Get a seed y of length $d = b(1 + \frac{\kappa}{2} + \log_2 \frac{n}{b})$ from the initially generated randomness, and

store it in local memory. We interpret y as $y = (y_0, y_1, \ldots, y_b) \in \{0,1\}^{(1+\frac{\kappa}{2})b} \times \left\{0, 1, 2, \ldots, \frac{n}{b} - 1\right\}^b$.

If there is theoretical knowledge for κ, then RRB provably (by Theorem 3.17 on page 78) extracts from $\lambda = \frac{32}{\kappa^2} \log_2 \frac{n}{\varepsilon}$ many independent samples from arbitrary (n,k)-sources, concatenated as a single input of length λn, for $b = \lambda \ln \frac{n}{\varepsilon}$, $\gamma = \frac{1-(1-\kappa)^{2/3}+\sqrt{1+8(1-\kappa)^{2/3}}}{8\ln(n/\varepsilon)}$, and $d = (\log_2 n + 2)b$. In this case, RRB extracts $m \geq n\big(1-(1-\kappa)^{2/3}\big)\big(1-(1-\kappa)^{1/3}\big)\kappa^{-2}\log_2 e = \Omega(n)$ bits that are ε-close to uniform, from λn bits input and d bits random seed. For instance, RRB can use a seed smaller than 6.7 MB to extract more than 8.8 GB of randomness for $n = 20$ GB, $\kappa = 0.5$ and $\varepsilon = 10^{-20}$.

3.3.5 Standard statistical tests.

Here we explain how to read the test results of NIST and DIEHARD, which are standard batteries for statistical tests of uniformity.

Each statistical test measures one property of the uniform distribution by computing a $\mathcal{P}$-value, which on ideal random inputs is uniformly distributed in $[0,1]$. For each NIST test, subsequences are derived from the input sequence and $\mathcal{P}$-values are computed for each subsequence. A significance level $\alpha \in [0.0001, 0.01]$ is chosen such that a subsequence passes the test whenever $\mathcal{P}$-value $\geq \alpha$ and fails otherwise. If we think that NIST is testing ideal random inputs, then the proportion of passing subsequences has expectation $1-\alpha$, and the *acceptable range of proportions* is the confidence interval chosen within 3 standard deviations. Furthermore, a second-order $\mathcal{P}$-value is calculated on the $\mathcal{P}$-values of all subsequences via a χ^2-test. An input passes one NIST test if (i) the input induces an acceptable proportion and (ii) the second-order $\mathcal{P}$-value ≥ 0.0001. An input passes one DIEHARD-test if $\mathcal{P}$-value is in $[\alpha, 1-\alpha]$.

We compare the statistical behavior of bits produced by our method with ideal random bits. For ideal random bit-sequences, α is the *ideal failure rate*. Anything significantly lower or higher than this indicates non-uniform input. In our tests, we choose the largest suggested significance level $\alpha = 0.01$; i.e. the hardest to pass the

test. All tests on our extracted bits appear statistically identical to ideal randomness (Table 3.1).

3.3.6 Experimental platform details.

The performance of the streaming RRB, von Neumann extractor, and Local Hash is measured on a desktop PC, with Intel Core i5 3.2 GHz CPU, 8 GB RAM, two 1 TB hard drives and kernel version Darwin 14.0.0. The performance of Trevisan's extractor is measured on the same PC with the entire input and intermediate results stored in main memory.

We use the following software platforms and libraries. TPIE [TPI13] is the C++ library on top of which we implement all streaming algorithms – TPIE provides application-level streaming I/O interface to hard disks. For arbitrary precision integer and Galois field arithmetic we use GMP [GMP14] and FGFAL [FGF07].

3.4 Experimental Data

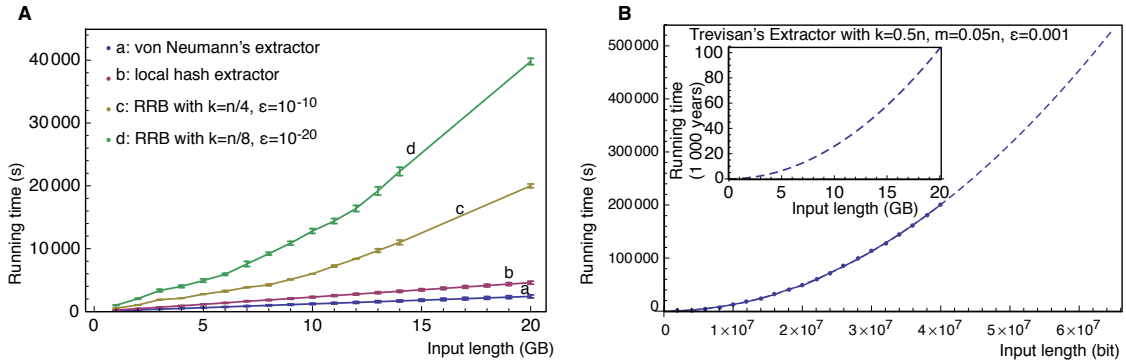

Figure 3-2: **Running time of RRB compared with other extractors.** Running time is measured on input samples of size 1 GB – 20 GB for von Neumann extractor, local hash extractor (with block-size 1024 bits), and for RRB (with $k = n/4, \varepsilon = 10^{-10}$ and $k = n/8, \varepsilon = 10^{-20}$). Trevisan's extractor is only measured for $\varepsilon = 0.001$ on samples of size up to 5 MB $= 4 \times 10^7$ bits, since the available implementations of finite fields cannot handle larger samples or smaller ε. The running time of Trevisan's extractor on larger input size (in particular, 103, 407 years for 20 GB input) is estimated by polynomial fitting assuming all data in the main memory, which is an unrealistic advantage. The exact form of the fittest polynomial is determined through cross-validation and standard analysis of polynomial norms.

Data category	NIST Test Suite			DIEHARD Test Suite	
	Number of tests	**Observed (Ideal) freq.**	$\mathcal{P}$**-val** < 0.0001	**Number of tests**	$\mathcal{P}$**-val outside** $(0.01, 0.99)$
Compressed audio	564	0 (1.52)	0 (0.06)	144	2 (2.88)
Compressed video	752	4 (2.03)	0 (0.08)	144	4 (2.88)
Compressed images	752	1 (2.03)	0 (0.08)	144	2 (2.88)
Compressed social network data	3008	9 (8.12)	0 (0.30)	576	10 (11.52)
Compressed DNA sequenced data	752	1 (2.03)	0 (0.08)	144	3 (2.88)
Compressed text	3008	11 (8.12)	0 (0.30)	576	14 (11.52)
Uncompressed audio	752	2 (2.03)	0 (0.08)	144	1 (2.88)
Uncompressed video	752	1 (2.03)	0 (0.08)	144	1 (2.88)
Uncompressed images	564	0 (1.52)	0 (0.06)	62	2 (1.24)
Uncompressed social network data	2256	5 (6.09)	1 (0.23)	432	5 (8.64)
Uncompressed DNA sequenced data	752	2 (2.03)	0 (0.08)	144	2 (2.88)
Uncompressed text	1504	0 (4.06)	1 (0.15)	288	7 (5.76)
Total	15416	36 (41.62)	2 (1.62)	2942	53 (58.84)

Table 3.1: **Details on empirically extracted bits versus ideal uniform random bits.** The first column is the tested data categories. The second column is the total number of NIST tests per data category. The third column is the number of NIST tests with a passing proportion outside the confidence interval (Methods p. 86). The fourth column is the number of NIST tests that the second-order $\mathcal{P}$-value is less than 0.0001. Both of them are compared with the expected statistic of NIST tests on ideal uniform random bits, which is listed in parenthesis. Similarly, the fifth column is the total number of DIEHARD tests and the sixth the number of failing tests (in parenthesis is the ideal failing number).

Extractor and data setting	NIST Test Suite		
	Number of tests	**Observed (Ideal) freq.**	**$\mathcal{P}$-val < 0.0001**
Raw data	2256	1931 (6.09)	1561 (0.22)
Von Neumann	2256	966 (6.09)	785 (0.22)
Von Neumann on adversary	2256	1507 (6.09)	1435 (0.22)
Local hash	2256	16 (6.09)	1 (0.22)
Local hash on adversary	2256	781 (6.09)	170 (0.22)
RRB	2256	4 (6.09)	0 (0.22)
RRB on adversary	2256	5 (6.09)	1 (0.22)

Table 3.2: **Comparative extraction quality performance.** The raw data consists of 12 files each of size 1000 MB from the 12 data categories and the adversarial data are generated by simply replacing 10 MB in each file with fixed values. NIST tests are applied on the raw data and extraction output of von Neumann, local hash, and RRB extractors on raw data and adversarial data. The second column is the total number of NIST tests per setting. The third column is the number of NIST tests that fail because of proportion (Methods p. 86), and the fourth column is the number of NIST tests that fail because of the second-order $\mathcal{P}$-value. All are compared with the expected number of the ideal uniform random bits. Except from RRB and "RRB on adversary", all other test results indicate non-uniform output (i.e. noticeably different from ideal uniform).

(a) Compressed audio

κ \ γ	1/2	1/4	1/8	1/16	1/32
1/2	0.1408	0.1330	0.1596	0.2660	0
1/4	0.1596	0.1596	0.1330	0.5319	0
1/8	0.1064	0.1330	0	0	0
1/16	0.2660	0.5319	0	0	0
1/32	0	0	0	0	0

(b) Audio

κ \ γ	1/2	1/4	1/8	1/16	1/32
1/2	0.1698	0.1164	0.1400	0.1182	0.1773
1/4	0.0887	0.0560	0.0532	0.1330	0.5319
1/8	0.0332	0	0	0	0
1/16	0	0	0	0	0
1/32	0	0	0	0	0

* Table entries are percentages; e.g. the first entry means 0.1408% = 0.001408.

(c) Compressed video

κ \ γ	1/2	1/4	1/8	1/16	1/32
1/2	0.1964	0.1272	0.1533	0.1330	0.1330
1/4	0.1850	0.1544	0.2013	0.1478	0.0887
1/8	0.1983	0.1869	0.2660	0.1773	0
1/16	0.2161	0.1182	0.1773	0.2660	0
1/32	0.2660	0.2660	0.3546	1.0638	0

(d) Video

κ \ γ	1/2	1/4	1/8	1/16	1/32
1/2	0.2579	0.2716	0.2482	0.2327	0.2660
1/4	0.2490	0.2992	0.3639	0.4137	0.5319
1/8	0.2660	0.3080	0.4255	0.5319	1.0638
1/16	0.1995	0.1773	0.1330	0	0
1/32	0.3546	0.3546	0	0	0

(e) Compressed image

κ \ γ	1/2	1/4	1/8	1/16	1/32
1/2	0.2872	0.3224	0.3919	0.2364	0.1773
1/4	0.2095	0.2128	0.2660	0.2660	0
1/8	0.2520	0.2128	0.3989	1.0638	0
1/16	0.1182	0	0	0	0
1/32	0	0	0	0	0

(f) Image

κ \ γ	1/2	1/4	1/8	1/16	1/32
1/4	0.4110	0.4074	0.3191	0.3324	0.4433
1/8	0.3622	0.3823	0.2520	0.3546	0.1773
1/16	0.3723	0.4199	0.3191	0.6649	0.5319
1/32	0.3657	0.4728	0.1330	0	0
1/64	0.4433	0.7092	0	0	0

(g) Compressed social network data

κ \ γ	1/2	1/4	1/8	1/16	1/32
1/2	0.2176	0.2037	0.2482	0.3657	0.3546
1/4	0.1924	0.1995	0.2520	0.4728	0.7092
1/8	0.2305	0.2240	0.2660	0.5319	0.5319
1/16	0.2327	0.2364	0.2660	0.5319	
1/32	0.0887	0	0	0	0

(h) Social network data

κ \ γ	1/2	1/4	1/8	1/16	1/32
1/8	0.3191	0.3137	0.3324	0.3989	0.2660
1/16	0.3103	0.3103	0.3682	0.4255	0
1/32	0.2128	0.3103	0.3191	0.5319	0
1/64	0.1330	0.1330	0	0	0
1/128	0	0	0	0	0

(i) Compressed DNA sequenced data

κ \ γ	1/2	1/4	1/8	1/16	1/32
1/2	0.1854	0.2150	0.1596	0.1995	0.2660
1/4	0.1471	0.1662	0.1120	0.1182	0
1/8	0.1418	0.1680	0.1064	0.1330	0
1/16	0.1662	0.2364	0	0	0
1/32	0.2660	0.3546	0	0	0

(j) DNA sequenced data

κ \ γ	1/2	1/4	1/8	1/16	1/32
1/8	0.2579	0.2660	0.1596	0	0
1/16	0.2240	0.1596	0	0	0
1/32	0.2364	0.2660	0	0	0
1/64	0	0	0	0	0
1/128	0	0	0	0	0

(k) Compressed text

κ \ γ	1/2	1/4	1/8	1/16	1/32
1/2	0.3707	0.3961	0.3191	0.2327	0.3546
1/4	0.2943	0.3158	0.3080	0.3546	0.7092
1/8	0.1950	0.2520	0.2128	0.2660	0
1/16	0.1662	0.2364	0.1330	0	0
1/32	0	0	0	0	0

(l) Text

κ \ γ	1/2	1/4	1/8	1/16	1/32
1/8	0.2812	0.2561	0.2240	0.2313	0.1064
1/16	0.2766	0.2660	0.2455	0.2837	0.0887
1/32	0.2402	0.2081	0.2128	0.2660	0.1773
1/64	0.1900	0.2128	0.1773	0.3546	0
1/128	0	0	0	0	0

Table 3.3: **Empirical estimation for κ and γ for the 12 data categories.** For every data category we realize the empirical estimation method described in Methods p. 84. Following the discussion in Section 3.3.3 on page 84, the selected acceptable (κ, γ) pair is the top-left corner of the consistent subtable where the percentage of failures is smaller than 0.25% (slightly below the ideal 0.27% for NIST). Samples used for parameter estimation are not used in any other part of the experiment.

Data category (compressed)	Audio		Video		Images		Social network		DNA sequenced		Text	
Sample size	1.5 GB		7.3 GB		4.8 GB		5 GB ×4		2.9 GB		12 GB ×4	
NIST Test Suite	**Prop.**	$\mathcal{P}$**-val**	**Prop.**	$\mathcal{P}$**-val**	**Prop.**	$\mathcal{P}$**-val**	**Prop.**	$\mathcal{P}$**-val**	**Prop.**	$\mathcal{P}$**-val**	**Prop.**	$\mathcal{P}$**-val**
Frequency	0.9914	0.090936	0.9853	0.474986	0.9960	0.334538	0.9923	0.048716	0.9880	0.047173	0.9927	0.022760
Block frequency	0.9914	0.075719	0.9853	0.213309	0.9933	0.001824	0.9910	0.129620	0.9907	0.048716	0.9863	0.042808
Cumulative sums	0.9957	0.213309	0.9853	0.401199	0.9960	0.155936	0.9913	0.070081	0.9913	0.307818	0.9928	0.020548
Runs	0.9943	0.719747	0.9907	0.275709	0.9920	0.075719	0.9903	0.021262	0.9947	0.062821	0.9900	0.062821
Longest run	0.9943	0.080519	0.9853	0.045675	0.9947	0.003851	0.9873	0.026948	0.9867	0.047173	0.9897	0.230755
Rank	0.9800	0.001399	0.9907	0.249284	0.9933	0.068999	0.9893	0.030806	0.9893	0.419021	0.9923	0.001296
FFT	0.9886	0.057146	0.9813	0.621506	0.9893	0.042808	0.9883	0.055361	0.9933	0.145326	0.9913	0.057146
Non-overlapping template	0.9898	0.000101	0.9899	0.011585	0.9900	0.003084	0.9897	0.000274	0.9901	0.001895	0.9903	0.000544
Overlapping template	0.9971	0.262249	0.9920	0.574903	0.9933	0.155936	0.9890	0.090936	0.9920	0.268917	0.9837	0.026948
Universal	0.9800	0.605916	0.9800	0.509162	0.9827	0.068999	0.9853	0.035174	0.9880	0.213309	0.9887	0.015598
Approximate entropy	0.9886	0.626709	0.9880	0.334538	0.9853	0.035174	0.9903	0.032923	0.9920	0.595549	0.9887	0.078086
Random excursions	0.9863	0.000533	0.9866	0.048716	0.9899	0.062821	0.9881	0.008016	0.9882	0.001949	0.9903	0.006483
Random excursions variant	0.9905	0.011250	0.9942	0.020548	0.9898	0.094664	0.9916	0.001738	0.9905	0.003201	0.9876	0.001001
Serial	0.9857	0.350485	0.9913	0.319084	0.9933	0.037566	0.9890	0.021999	0.9880	0.034031	0.9905	0.062821
Linear complexity	0.9857	0.021262	0.9947	0.202268	0.9867	0.549331	0.9893	0.181557	0.9853	0.008491	0.9913	0.129620

Data category (uncompressed)	Audio		Video		Images		Social network		DNA sequenced		Text	
Sample size	9GB		3.7 GB		3.8 GB		5 GB ×3		8 GB		20 GB ×2	
NIST Test Suite	**Prop.**	$\mathcal{P}$**-val**	**Prop.**	$\mathcal{P}$**-val**	**Prop.**	$\mathcal{P}$**-val**	**Prop.**	$\mathcal{P}$**-val**	**Prop.**	$\mathcal{P}$**-val**	**Prop.**	$\mathcal{P}$**-val**
Frequency	0.9960	0.060875	1.0000	0.073417	0.9907	0.022760	0.9911	0.105618	0.9880	0.554420	0.9927	0.035174
Block frequency	0.9920	0.699313	0.9867	0.595549	0.9820	0.514124	0.9942	0.030297	0.9880	0.045675	0.9940	0.027868
Cumulative sums	0.9940	0.023545	0.9987	0.204985	0.9953	0.055361	0.9898	0.096578	0.9853	0.358641	0.9917	0.099513
Runs	0.9907	0.474986	0.9960	0.437374	0.9973	0.081760	0.9884	0.093720	0.9907	0.068999	0.9920	0.268917
Longest run	0.9947	0.075719	0.9947	0.108791	0.9920	0.262249	0.9871	0.034031	0.9893	0.002512	0.9873	0.176657
Rank	0.9867	0.678666	0.9907	0.026948	0.9880	0.092319	0.9898	0.016149	0.9933	0.073417	0.9827	0.089572
FFT	0.9840	0.153763	0.9867	0.383827	0.9880	0.304126	0.9893	0.003712	0.9800	0.042118	0.9847	0.149495
Non-overlapping template	0.9897	0.000195	0.9899	0.003996	0.9898	0.004462	0.9900	0.000070	0.9901	0.002559	0.9904	0.000057
Overlapping template	0.9840	0.000216	0.9813	0.569766	0.9920	0.272297	0.9867	0.122325	0.9827	0.153763	0.9827	0.045675
Universal	0.9907	0.423545	0.9971	0.042808	0.9960	0.779188	0.9867	0.075719	0.9853	0.135331	0.9880	0.118812
Approximate entropy	0.9907	0.194289	0.9933	0.075719	0.9880	0.066882	0.9902	0.102526	0.9933	0.451234	0.9947	0.042808
Random excursions	0.9872	0.005930	0.9863	0.030806	0.9931	0.103401	0.9867	0.003577	0.9883	0.025522	0.9913	0.001691
Random excursions variant	0.9885	0.010817	0.9881	0.017912	0.9920	0.007616	0.9900	0.000895	0.9901	0.009509	0.9919	0.004681
Serial	0.9900	0.282626	0.9880	0.124115	0.9873	0.275709	0.9920	0.081760	0.9867	0.414525	0.9900	0.010606
Linear complexity	0.9920	0.162606	0.9813	0.275709	0.9880	0.075719	0.9920	0.191687	0.9947	0.059923	0.9933	0.213309

Table 3.4: **Extraction quality evaluation by NIST.** The input samples to RRB are files of size 1.5 GB – 20 GB from 12 data categories, whenever there is no large enough file we concatenate (potentially correlated) smaller files together. Sample sizes are listed for every data category and RRB is applied independently on every input sample with parameters estimated in Table 3.3. For every run of RRB the extraction quality is evaluated by the NIST test suite for 4 times with the test input equally partitioned into 50, 100, 200, and 400 subsequences respectively. NIST tests compute $\mathcal{P}$-values for every subsequence in every test and then apply a χ^2 test to calculate a second-order $\mathcal{P}$-value. The overall proportion of subsequences with $\mathcal{P}$-value ≥ 0.01 and the *minimum* second-order $\mathcal{P}$-values are reported. To pass one NIST test, the second-order $\mathcal{P}$-value should be greater than 0.0001, and the proportion of subsequences with $\mathcal{P}$-value ≥ 0.01 must be greater than 0.940, 0.960, 0.965, and 0.975 respectively for 50 – 400 subsequences. The second-minimum $\mathcal{P}$-value of "non-overlapping template" tests is 0.001895 for uncompressed social data and 0.003511 for uncompressed text. The reported $\mathcal{P}$-value is *not* uniformly distributed, since it is the minimum among multiple NIST tests.

Data category (compressed)	Audio	Video	Images	Social network	DNA sequenced	Text
DIEHARD Suite	$\mathcal{P}$-val	$\mathcal{P}$-val	$\mathcal{P}$-val	$\mathcal{P}$-val	$\mathcal{P}$-val	$\mathcal{P}$-val
Birthday spacings	0.438088	0.310881	0.058193	0.554324	0.808198	0.000052
Overlapping permutations	0.752959	0.428180	0.639592	0.723372	0.056572	0.243835
Ranks of 31x31 matrices	0.569184	0.339979	0.920656	0.482406	0.385041	0.362155
Ranks of 32x32 matrices	0.547909	0.403446	0.613285	0.433728	0.352847	0.321909
Ranks of 6x8 matrices	0.645499	0.845557	0.345142	0.018670	0.701125	0.881740
Bit stream test	0.387205	0.593575	0.014402	0.654673	0.125810	0.431669
Monkey tests OPSO	0.355216	0.197500	0.667192	0.914133	0.238029	0.572953
Monkey tests OQSO	0.645485	0.669454	0.480169	0.629274	0.027371	0.834129
Monkey tests DNA	0.486269	0.940092	0.174359	0.591367	0.874612	0.858750
Count 1's in a stream of bytes	0.608089	0.917741	0.427466	0.822289	0.726707	0.597594
Count 1's in specific bytes	0.714184	0.097330	0.151793	0.462850	0.294159	0.656062
Parking lot test	0.152996	0.151125	0.687893	0.119339	0.251395	0.044892
Minimum distance test	0.897483	0.896119	0.638546	0.846002	0.654949	0.466783
Random spheres test	0.546443	0.057325	0.295651	0.501201	0.782601	0.678879
The squeeze test	0.541244	0.436086	0.144311	0.721372	0.942856	0.919760
Overlapping sums test	0.240819	0.239792	0.053650	0.577147	0.055292	0.307435
Runs test (up)	0.962195	0.537571	0.269606	0.671718	0.411167	0.079167
Runs test (down)	0.813418	0.358613	0.321261	0.837789	0.472629	0.807418
Craps test no. of wins	0.642862	0.586490	0.113479	0.737114	0.252371	0.015010
Craps test throws per game	0.276496	0.697953	0.028449	0.927459	0.278761	0.611609

Data category (uncompressed)	Audio	Video	Images	Social network	DNA sequenced	Text
DIEHARD Suite	$\mathcal{P}$-val	$\mathcal{P}$-val	$\mathcal{P}$-val	$\mathcal{P}$-val	$\mathcal{P}$-val	$\mathcal{P}$-val
Birthday spacings	0.385522	0.828963	0.004437	0.208048	0.057008	0.955554
Overlapping permutations	0.724376	0.512534	0.985741	0.209806	0.863579	0.918563
Ranks of 31x31 matrices	0.842685	0.636229	0.357475	0.881277	0.600983	0.409768
Ranks of 32x32 matrices	0.822334	0.503910	0.890231	0.619142	0.419850	0.576953
Ranks of 6x8 matrices	0.591588	0.563394	0.957945	0.013593	0.175439	0.339026
Bit stream test	0.155471	0.321259	0.228977	0.527317	0.863584	0.206301
Monkey tests OPSO	0.630120	0.714526	0.012538	0.058461	0.534801	0.723825
Monkey tests OQSO	0.398779	0.380577	0.955060	0.251407	0.694832	0.169836
Monkey tests DNA	0.862675	0.260406	0.576408	0.890297	0.507449	0.823096
Count 1's in a stream of bytes	0.452006	0.915969	0.211816	0.616283	0.206299	0.192805
Count 1's in specific bytes	0.283858	0.819726	0.257865	0.244149	0.763107	0.782839
Parking lot test	0.020669	0.256870	0.590298	0.293398	0.931840	0.076152
Minimum distance test	0.467898	0.851627	0.267924	0.343415	0.767186	0.734031
Random spheres test	0.316626	0.077993	0.473525	0.700093	0.822260	0.033127
The squeeze test	0.922929	0.258466	0.260324	0.584267	0.434682	0.751415
Overlapping sums test	0.001460	0.030426	0.074745	0.009624	0.034810	0.080606
Runs test (up)	0.185805	0.954631	0.998131	0.516886	0.526526	0.637137
Runs test (down)	0.034008	0.365320	0.967070	0.368460	0.390886	0.759504
Craps test no. of wins	0.032640	0.979268	0.068112	0.537164	0.856300	0.259566
Craps test throws per game	0.018952	0.675906	0.882955	0.428987	0.757852	0.280388

Table 3.5: **Extraction quality evaluation by DIEHARD.** The test inputs for DIEHARD are the same extracted outputs as in Table 3.4. DIEHARD computes $\mathcal{P}$-values for every extracted output in every statistical test. When there are multiple $\mathcal{P}$-values in one test DIEHARD uses a Kolmogorov-Smirnov (KS) test to obtain a final $\mathcal{P}$-value. In case there are multiple final $\mathcal{P}$-values the first obtained one is reported. The test passes if the final $\mathcal{P}$-value is in $[0.01, 0.99]$.

Description	Url	Remark
Social network (compressed and uncompressed)		
Stackoverflow data dump	https://archive.org/download/stackexchange	Last download: 2014-09-26 / plain text
Amazon product co-purchasing network metadata	https://snap.stanford.edu/data/amazon-meta.html	Product meta data and reviews / plain text
Text (compressed and uncompressed)		
Linguistic Data Consortium (LDC). Data sources: varied, web collection,telephone speech, newswire news magazine, journal articles, continuous speech corpus	https://catalog.ldc.upenn.edu	Plain text (untagged) and text/words tagged with statistics DVDs: 1–6
Images (compressed and uncompressed)		
Astronomy picture of the day	http://apod.nasa.gov/apod/astropix.html	Last download: 2014-9-13 / (original) jpg From: abell1185_cfht_big.jpg To: zvezdalaunch_nasa.jpg
Audio (compressed and uncompressed)		
Project Gutenberg	http://www.gutenberg.org/browse/categories/1	Last download: 2014-12-14 / (original) m4b All
DNA sequences (compressed and uncompressed)		
Assembled human genomes from NCBI	ftp://ftp.ncbi.nih.gov/genomes/H_sapiens /Assembled_chromosomes/	From: hs_alt_CHM1_1.1_chr10.fa.gz To: hs_alt_HuRef_unplaced.mfa.gz

Video (compressed and uncompressed) From: https://www.youtube.com		
Video title	Video title	Video title
A Bridge Too Far 1977 - Full Movie	Alfred Hitchcock - Notorious	Don't Come Knocking 2005 - Wim Wenders
Fear and Desire (Restored Full Length) (1953) - Kubrick	Hercules	Star Wars Threads of Destiny (2014)
The Complete Citizen Kane - Documentary	The Phantom Edit	Waterloo (1970)
Zulu (1964) PL	David And Goliath - Orson Welles	Inside the Making of Dr. Strangelove
Jerry Seinfeld - I'm telling you for the last time	laberinto de pasiones 1982 - Pedro Almodovar	Live concert Rachmaninov - Vespers
Manu Chao - Clandestino Full Album HD	Manos Hatzidakis (A walk on the moon)	Pirates of the Caribbean Soundtrack
Ramones - Live In Cologne, Gemany 92 full concert	Rimsky-Korsakov Sheherazade - J.Suk, Czech PO, A.Rahbari	Seinfeld How It Began (full)
Star Trek Friendship One	Star Trek full episode season 3 episode 20	Star Trek The Enterprise Incident
Star Trek Mirror season 2 episode 10	Star Trek the devil in the dark season 1 episode 26	Star Trek Voyager - The Borg Arc
The Best of Debussy	The Sex Pistols Live At Winterland San Fransisco 1978	Until The End Of The World - Wim Wenders
Woody Allen - Wild Man Blues (1997)	Roman Market (Manos Hatzidakis)	Amélie - Full Soundrack
A mi madre le gustan las mujeres	Charlie Chaplin The Kid - Film 1921	Citizen Kane Anatomy Of A Classic

Table 3.6: **Data used in experiments.**

Chapter 4

Limits of the Multi-Stream Model for Random and Pseudorandom Objects

In this chapter, we devise a new lower bounding methodology (Section 4.1), which is one of the central technical contributions of this dissertation. With this methodology, we first show impossibility of cryptographic primitives in Section 4.2, which complements the constructions of streaming pseudorandom generators and PKE schemes in Section 2.4 (page 39) and Section 2.5 (page 44). Then, we demonstrate lower bounds for randomness extractors in Section 4.3 that complements the construction of streaming extractors in Section 3.1 (page 57).

4.1 Overview of the Lower-Bound Technique

In this section, we first describe the lower bound technique.

Dependency graphs and dependency trees. The concepts of dependency graphs and dependency tree were originally introduced in [BH12] based on the treatment in [GHS09, GS05], where they discuss the sortedness across different blocks. We make use of these concepts in a quite different way, with an information theoretical argu-

ment on the growth of the entropy inside every single block.

Let an (s, t, p) streaming algorithm have $p + 1$ phases induced by the p passes: whenever one pass ends on any of the streams the computation enters a new phase. The crucial observation in [GS05] is that when writing to a particular cell in the i-th phase, what is written only depends on the local memory together with the t cells currently being scanned by the t heads. Moreover, those t cells are written before the i-th phase, since no cell can be visited twice in a single phase.

Definition 4.1. *Let F be a deterministic streaming algorithm such that on input x, F makes at most p passes over t external tapes. The* dependency graph*, denoted by $\Gamma(x)$, is a directed $(p+1)$-layered graph associated with the computation of $F(x)$ as follows. Level i is associated with the i-th layer in $\Gamma(x)$ and it consists of all nodes labeled (v, i) if and only if the tape cell v has ever been visited in the i-th phase or before.*[1] *$\Gamma(x)$ has an edge $(u, i) \to (v, i+1)$ if and only if any head is reading the cell u when v is being written in the $(i+1)$-st phase. Furthermore, there is always an edge $(u, i) \to (u, i+1)$ as long as (u, i) is in $\Gamma(x)$ and $i \leq p$. The* dependency tree *rooted at v is the subgraph of all nodes in $\Gamma(x)$ with a directed path to $(v, p+1)$.*

In the dependency graph $\Gamma(x)$, each level represents a single phase in the computation of $F(x)$. Therefore the nodes (except for those at level 1) have in-degree at most t, while all edges are heading to the next level. Intuitively, those directed edges depict the information flow excluding the local memory.

We also remark that not all old passes (over the remaining streams) are necessarily finished when a new phase begins. In the new phase the algorithm will continue to process old passes.

Example (dependency graph) An example of a dependency graph is depicted on Figure 4-1, which is an example of a simple algorithm that adds up 11100 and 110. This dependency graph has two external tapes and seven levels, where each level consists of exactly the nodes corresponding to all tape cells that have ever been visited before the associated pass begins.

[1] For completeness we assume that all input cells are visited before the first phase.

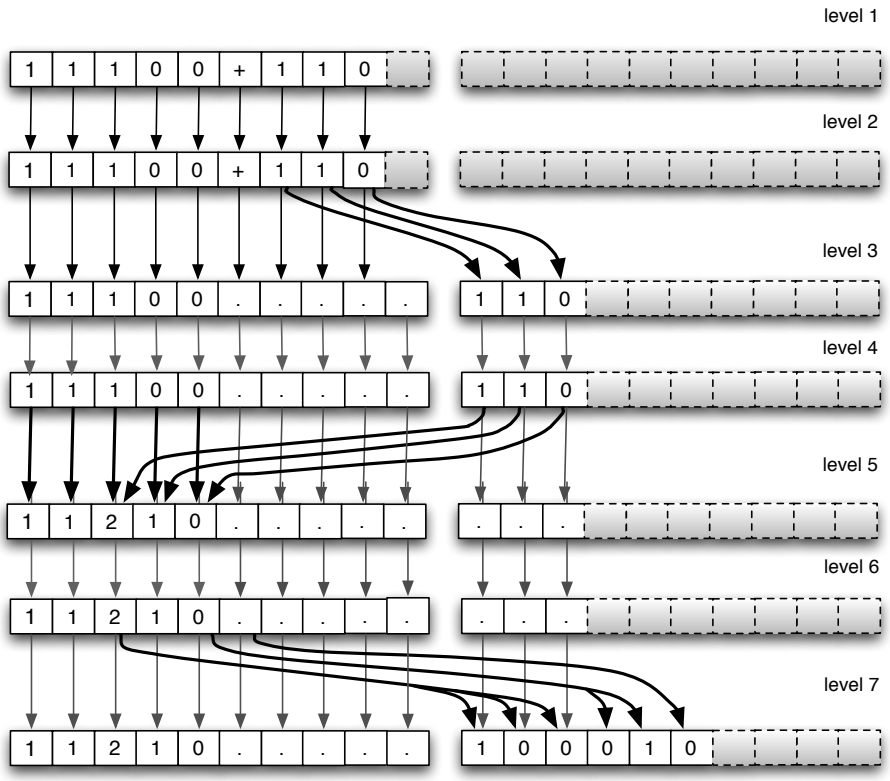

Figure 4-1: **A dependency graph example.**

In the dependency graph, level 1 represents the input "11100" and "110"; level 3 corresponds to copying the second input "110" to the second tape; level 5 adds up the two numbers "11100" and "110" without carries; level 7 deals with carries. Level 2,4 and 6 are there only for completeness, since every time we begin two passes simultaneously, and as a result the actual task of passes associated with level 2,4 and 6 are handled in level 3,5 and 7 respectively.

The first six levels are trivial. In the last level, the algorithm maintains a counter in its working memory to count how many cells have been scanned before finding the next "0" or "2", and it writes $100\cdots0$ if reads "2" or $011\cdots1$ if reads "0", where the length of $100\cdots0$ (resp. $011\cdots1$) equals to the value stored in the counter. □

Every node v in the dependency graph defines a sub-tree[2] rooted at v with all leaves at level 1. This tree is called the *dependency tree of* v. Intuitively, it describes the information flow, excluding the working memory, from input cells "collecting

[2] If two distinct parent nodes share the same child u we create two copies of u including the subtrees rooted at u.

information" at v. The information going to v through its internal working memory is upper bounded by the size of internal space s and the number of internal nodes in the dependency tree.

Figure 4-2 is a dependency tree example for the root v being the first output cell in Figure 4-1. (a) is the skeleton of the dependency tree rooted at v; (b) is the actual dependency tree of v. (b) duplicates at level 4 to eliminate the shared children and to transform the skeleton to a tree, since there is a shared children node at level 4 in the skeleton. In particular, Lemma 4.3 and Corollary 4.4 hold for the skeleton as well as for the corresponding dependency tree.

Then, we generalize the definition of dependency trees/graphs for blocks. The *blocks* are "super nodes" in $\Gamma(x)$ by merging nodes with the same dependency.

Definition 4.2. *A* block *is an equivalence class consisting of all nodes corresponding to tape cells at the same level on the same tape such that they depend on exactly the same set of blocks at the previous level. Specifically, an* input block *refers to a set of nodes at the first level corresponding to consecutive tape cells on the input tape.*

Note that if two cells, from the same tape and the same level, have the same dependency on blocks at the previous level, then any cell in between must have exactly the same dependency, because the dependency changes "monotonically". Therefore, the partition of blocks is well-defined such that every block consists of only consecutive tape cells.

Henceforth, we abuse notation and let $\Gamma(x)$ be the generalized dependency graph whose nodes corresponds to blocks as in Definition 4.2. Intuitively, blocks are used to package the entropy from the input, and $\Gamma(x)$ describes the direct information flow during the computation, i.e. except from those bits stored and transferred in the local memory. Later on in the proofs, we formalize the idea to analyze statistical dependencies between the input and the output with dependencies of blocks in $\Gamma(x)$.

We bound the number of blocks at each level for the input x partitioned into b input block, i.e. $x = (x_1, x_2, \cdots, x_b)$.

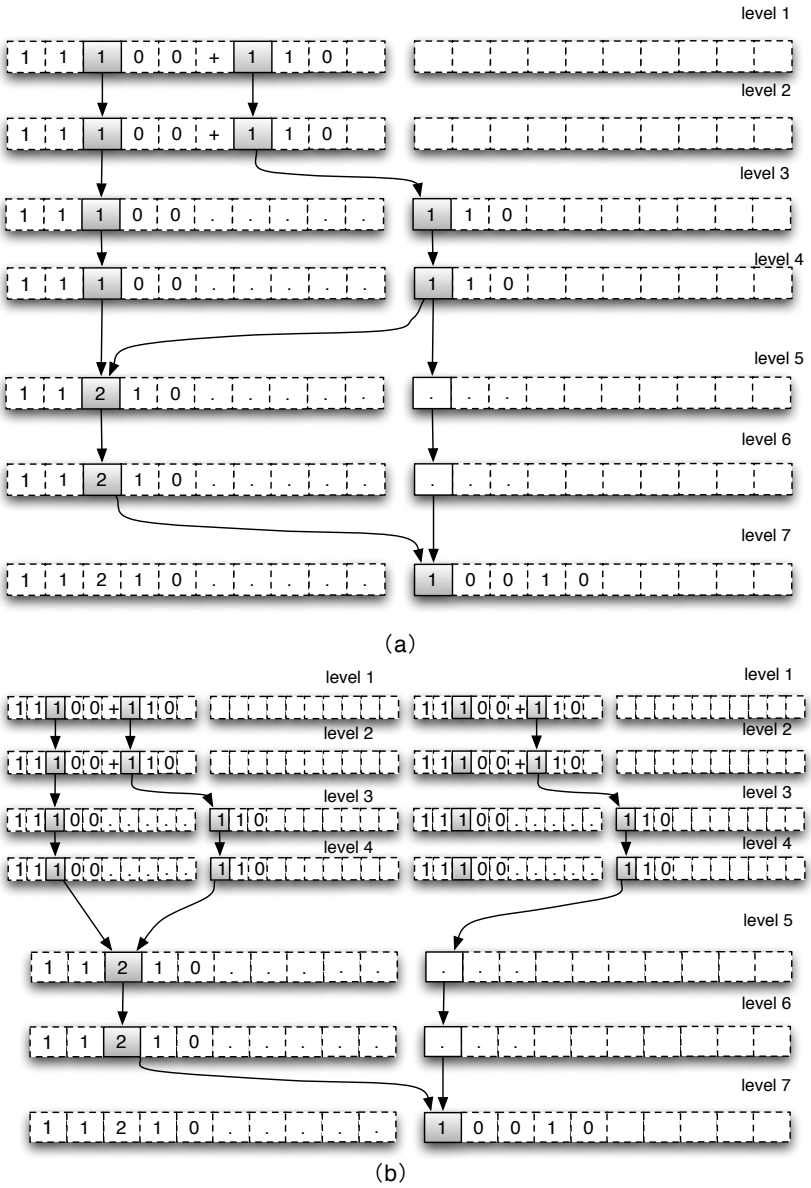

Figure 4-2: **A dependency tree example.**

Lemma 4.3 (cf. Lemma 3.4 in [BH12])**.** *Suppose x is partitioned into b blocks for $b \geq 1$. For a cell v, we write $\mathcal{I}_b(v) \subseteq \{1, 2, \cdots, b\}$ to denote the set of input blocks that v depends on according to its dependency tree, and let $r(v)$ and $l(v)$ denote the cell immediately to the right and the the left of v, respectively. Let C be the set of cells on any single tape at level i in $\mathcal{G}(x)$, for $1 \leq i \leq p+1$. Let the set $S_r = \{v \in C | \mathcal{I}_b(v) \neq \mathcal{I}_b(r(v))\ \}$ and $S_l = \{v \in C | \mathcal{I}_b(v) \neq \mathcal{I}_b(l(v))\ \}$. Then, $|S_r| = |S_l| \leq (b+1)t^{i-1}$.*

Corollary 4.4. *Partition x into b input blocks and let $\Gamma(x)$ be the dependency graph. Then, the number of blocks at level i in $\Gamma(x)$ is bounded $\leq (b+1)t^{i-1}$, where t denotes the number of tapes.*

Proof of Lemma 4.3. The first part $|S_r| = |S_l|$ is obvious by the symmetry in the definition of S_r and S_l. We show the upper bound by induction on i. For $i = 1$, S_r consists of all cells at the right boundaries of each block x_j for $1 \leq j \leq b$ and the immediate cell to the left of x, $|S_r| = b + 1$.

For $i > 1$, note that every element $v \in S_r$ must injectively correspond to some cell u at level $i-1$ such that $\mathcal{I}_b(u) \neq \mathcal{I}_b(r(u))$. Because $\mathcal{I}_b(v) \neq \mathcal{I}_b(r(v))$ implies that after written to the cell v and before writing to $r(v)$, at least one head moves forward and crosses a block boundary at level $i-1$. By induction hypothesis, there are at most $(b+1)t^{i-2}$ such u in each tape at level $i-1$, and in total t tapes, so that $|S_r| \leq (b+1)t^{i-1}$. □

4.2 Lower Bounds for Cryptographic Primitives

4.2.1 Lower Bound for Pseudorandom Generators

The lower bound technique is first applied to prove that super-linear stretch pseudorandom generators cannot be computed with constant many passes in the multi-stream model. Note that for every positive constant $c > 0$ and any sub-linear n^{1-c} stretch is easy to achieve, by simply running in parallel n^{1-c} copies of a single-bit stretch pseudorandom generator on independent seeds of length n^c.

Theorem 4.5. *Suppose $\ell(n) = \omega(n)$ and $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ is a pseudorandom generator. Then, no streaming algorithm can compute G.*

We prove Theorem 4.5 by analyzing the information flow in the computation, and by partitioning appropriately the output into blocks, we upper bound the entropy transferred to each output block from the input. Intuitively, a single block cannot collect much entropy and therefore it cannot induce large stretch.

Our proof makes use of the following observation and the concept of a dependency graph originally introduced in [BH12, GHS09] (in fact, [GS05]). We tailor them for cryptographic applications to partition the computation into $p + 1$ phases corresponding to p passes.

Observation 4.6 (cf. [GS05])**.** *When a tape cell is written, its content only depends on the local memory and the t cells currently being scanned by the heads of the external streams. Moreover, those t cells are written before this pass, since no cell can be visited twice before making a new pass.*

Proof of Theorem 4.5. Suppose G is efficiently computable in the streaming model, and the input to G is $x \in_R \{0,1\}^n$. We want to distinguish $G(x)$ from $y \leftarrow \mathcal{U}_{\ell(n)}$. First, we exhibit an advised distinguisher as follows, where the advice A is a set of strings depending merely on the input length.

Input: 1^n and y
Data: Advice A
for $z \in A$ **do**
  if z *appears in* y **then**
    Output: 1
  end
end
Output: 0

Algorithm 3: $\mathcal{D}_A$ – distinguishing $G(x)$ from y (with advice A)

To illustrate $\mathcal{D}_A$ is an efficient distinguisher, the following claim suffices.

Claim 4.7. *There is an advice A satisfying the following two conditions:*

i. $\Pr[\mathcal{D}_A\big(G(\mathcal{U}_n)\big) = 1] - \Pr[\mathcal{D}_A(\mathcal{U}_{\ell(n)}) = 1] \geq 1/n^{\mathrm{O}(1)}$

ii. A has at most $\mathsf{poly}(n)$ *many elements.*

Now, we show how to determine a useful A (later on we will show how to generate it). Consider the computation process of G on input x and the corresponding dependency graph $\mathcal{G}(x)$. The input x, written on the first stream, is partitioned into $b = \lceil \frac{n}{\log n} \rceil$ input blocks, with at most $\log n$ bits in each block. Here each input block must contain only consecutive bits of x from consecutive tape cells by Definition 4.2. At the end of the computation, when the output $G(x)$ is written on the output tape, $G(x)$ has at most $(b+1)t^p$ blocks by Corollary 4.4. Without loss of generality, we assume that there are exactly $(b+1)t^p$ blocks, labeled from 1 to $(b+1)t^p$ according to their positions.

Recall that $|G(x)| = \ell(n)$, each block in the $G(x)$ has $\frac{\ell(n)}{(b+1)t^p}$ bits on the average. Therefore, there must be a block labeled v that outputs, in expectation, at least $\frac{\ell(n)}{(b+1)t^p}$ bits when $x \leftarrow \mathcal{U}_n$. Let $G(x)|_v$ denote the output of $G(x)$ in the block v. Since $|G(x)| = \ell(n)$, it always holds $\big|G(x)|_v\big| \leq \ell(n)$, and therefore,

$$\Pr_x \left[\big|G(x)|_v\big| > \frac{\ell(n)}{2(b+1)t^p} \right] > \frac{1}{2(b+1)t^p} \tag{4.1}$$

Let the advice A consist of all possible strings that are sufficiently long and could appear as $G(x)|_v$. Formally,

$$A = \left\{ z \Big| \exists x \in \{0,1\}^n \text{ such that } z = G(x)|_v \text{ and } |z| > \frac{\ell(n)}{2(b+1)t^p} \right\} \tag{4.2}$$

By (4.1) and (4.2), $G(x)|_v \in A$ with probability greater than $\frac{1}{2(b+1)t^p}$. Therefore,

$$\Pr[\mathcal{D}_A\big(G(\mathcal{U}_n)\big) = 1] \geq \Pr[G(x)|_v \in A] > \frac{1}{2(b+1)t^p} = \Omega\left(\frac{\log n}{t^p \cdot n}\right) \tag{4.3}$$

However, for a uniformly distributed input,

$$\begin{aligned}\Pr[\mathcal{D}_A(\mathcal{U}_{\ell(n)})=1] &\le \sum_{z\in A}\Pr[z \text{ appears in } \mathcal{U}_{\ell(n)}]\\ &\le \sum_{z\in A}\sum_{j=1}^{\ell(n)}\Pr[z \text{ appears in } \mathcal{U}_{\ell(n)} \text{ starting at position } j]\\ &\le \sum_{z\in A}\sum_{j=1}^{\ell(n)}\frac{1}{2^{|z|}} < \sum_{z\in A}\sum_{j=1}^{\ell(n)}2^{-\frac{\ell(n)}{2(b+1)t^p}}\\ &= 2^{-\frac{\ell(n)}{2(b+1)t^p}}\ell(n)|A| = 2^{\log\ell(n)-\frac{\ell(n)}{2(b+1)t^p}}|A| \qquad (4.4)\end{aligned}$$

As long as $\ell(n)=\omega(n)$ and $t,p=\mathrm{O}(1)$, and recalling $b=\Theta(n/\log n)$, there is

$$\frac{\ell(n)}{2(b+1)t^p}=\Omega\left(\frac{\ell(n)\log n}{n}\right)=\omega(\log\ell(n))=\omega(\log n)$$

Thus, (4.4) implies

$$\Pr[\mathcal{D}_A(\mathcal{U}_{\ell(n)})=1] < 2^{-\omega(\log\ell(n))}|A| = n^{-\omega(1)}|A| \qquad (4.5)$$

Combining (4.3) and (4.5), the advantage of $\mathcal{D}_A$ is non-negligible when $|A|=n^{\mathrm{O}(1)}$.

$$\Pr[\mathcal{D}_A\big(G(\mathcal{U}_n)\big)=1]-\Pr[\mathcal{D}_A(\mathcal{U}_{\ell(n)})=1]=\Omega\left(\frac{\log n}{t^p\cdot n}\right)-n^{-\omega(1)}=\Omega\left(\frac{\log n}{t^p\cdot n}\right)$$

Therefore, it suffices to show the advice A has polynomial size $|A|=n^{\mathrm{O}(1)}$, which finishes the proof of Claim (4.7). A trivial upper bound appears to be the total number of all possible outputs in block v. We use this bound and calculate how many distinct strings can appear as $G(x)|_v$.

We analyze how $G(x)|_v$ is determined in $\mathcal{G}(x)$. Note that the content of a block is fully determined by: (i) the blocks with an edge to it in the dependency graph; (ii) and a snapshot of the local memory when the head moves into it. It easily generalizes to that $G(x)|_v$, as the content in block v, is fully determined by all input blocks corresponding to leaf nodes in v's dependency tree together with the snapshots for all non-leaf nodes.

Recall that in the dependency graph $\mathcal{G}(x)$ every node has in-degree at most t and the graph has p layers in total. Thus, we can bound the size of the dependency tree, i.e. t-branching and height p. Since all the tape cells in block v share the same dependency tree and input dependency set by our partitioning method, we assert that $G(x)|_v$ depends on at most t^p input blocks, each with length $\log n$ at most. On the other hand, the dependency tree has at most $1 + t + \cdots + t^{p-1}$ non-leaf nodes, each of which depends on an s-bit snapshot of the local memory besides other blocks directing to it.

Then, the entropy of $G(x)|_v$ is upper bounded as $\mathcal{H}\big(G(x)|_v\big) \leq t^p \times \log n + (1 + t + \cdots + t^{p-1})s$. Plugging in that $t = \mathrm{O}(1), p = \mathrm{O}(1), s = \mathrm{O}(\log n)$, this concern becomes

$$\mathcal{H}\big(G(x)|_v\big) \leq t^p \times \log n + (1 + t + \cdots + t^{p-1})s = t^p \log n + \frac{t^p - 1}{t - 1}\mathrm{O}(\log n) = \mathrm{O}(\log n)$$

As a result, $G(x)|_v$ has at most $2^{\mathrm{O}(\log n)} = n^{\mathrm{O}(1)}$ possibilities, which implies $|A| = n^{\mathrm{O}(1)} = \mathsf{poly}(n)$.

Thus, we have proved Claim (4.7) and it follows that $\mathcal{D}_A$ is an efficient distinguisher.

It remains to to argue that the advice is polynomial time computable, in order to eliminate the necessity of requiring a non-uniform oracle advice. It suffices to enumerate all possible computation of block v.

To enumerate the computation process of block v on every input, we do follows:

i. $A' \leftarrow \emptyset$;

ii. enumerate all dependency trees of height p where every node has in-degree either t or 1;

iii. for all dependency trees, enumerate all possible values in leaf nodes, each has $\log n$ bits;

iv. for every tree with leaf nodes, enumerate the local memory when entering each of its blocks;

v. from leaf nodes to the root node evaluate the content of every node;

vi. place the whole string in block v to A' if it has length greater than $\frac{\ell(n)}{2(b+1)t^p}$;

vii. output A'.

A' is not exactly the same as A defined in (4.2), for A' may have more elements than A. However, since $A \subseteq A'$, this advice A' does satisfy Claim (4.7). As a result, the distinguisher defined by A' is good enough.

Therefore, we have found a uniform polynomial time efficient distinguisher $\mathcal{D}$ for G, violating the assumption of G being a pseudorandom generator. □

4.2.2 Lower Bound for Public-Key Encryption

Theorem 4.8. *For every IND-CPA secure PKE whose decryption scheme is a streaming algorithm, the private key has length $\Omega(N)$, where N is the length of the plaintext.*

Proof of Theorem 4.8. We begin with an overview of the proof.

We use the idea of dependency graph introduced in Section 4.1 and the notion of blocks as in Definition 4.2 (page 98). By the way of contradiction assume that the private key has length $n = \mathrm{o}(N)$.

First, partition the length n private key $\mathcal{SK}$ into blocks of size at most $\log N$ and fix the other part of the input. Then, the outputs of the decryption scheme are also partitioned into blocks according to their dependency on blocks in $\mathcal{SK}$. With similar argument as in Section 4.2.1, we obtain that each output block has at most $\mathrm{O}(\log N)$ bits entropy. This is because each output block depends on $\mathrm{O}(1)$ many input blocks each of size $\mathrm{O}(\log N)$, and also it depends on $\mathrm{O}(1)$ many snapshots of the local memory corresponding to non-leaf nodes each of size $\mathrm{O}(\log N)$.

Since $n = \mathrm{o}(N)$, there are at most $\mathrm{O}(n/\log N) = \mathrm{o}(N/\log N)$ output blocks by Corollary 4.4 on page 100. Recalling that output length is N, every output block carries $\omega(\log N)$ bits in expectation. In particular, there must be an output block B expected to output $\omega(\log N)$ bits.

However, note that B has entropy at most $\mathrm{O}(\log N)$. So, it is possible to enumerate the $\mathrm{O}(\log N)$ bits entropy and thus decrypt B with probability at least

$2^{-\mathrm{O}(\log N)} = N^{-\mathrm{O}(1)}$, which is significantly larger than a random guessing for $\omega(\log N)$ bits.

Here the formal argument starts.

We make exactly the same contradiction hypothesis as in the overview. First, for a fixed public-key system with public key $\mathcal{PK}$ and length-n private key $\mathcal{SK}$, we pick a plaintext $\mathbf{x} \in_R \{0,1\}^N$. We also sample[3] $\mathbf{c} = \mathbf{Enc}_{\mathcal{PK}}(\mathbf{x}, \mathbf{r}) \in \mathbf{Enc}_{\mathcal{PK}}(\mathbf{x})$ as the ciphertext of $\mathbf{x}$, i.e. $\mathbf{c}$ is sampled from the ciphertext space where $\mathbf{x} \leftarrow \mathcal{U}_N$, and $\mathbf{r}$ is the uniform local random bits used by the encryption scheme. Now, let us consider the dependency graph of $\mathbf{Dec}_{\mathcal{SK}}(\mathbf{c})$. Suppose $B = B(\mathbf{c})$ is an output block with maximal expected output length $\ell_B = \mathbf{E}[\ell_B(\mathbf{c})] = \mathbf{E}[|B(\mathbf{c})|] = \mathrm{O}\left(\frac{N}{n}\log N\right) = \omega(\log N)$.

The adversary randomly chooses two message $\mathbf{x}_0, \mathbf{x}_1 \in_R \{0,1\}^N$, and transmits them to the challenge oracle. The challenge oracle will select one out of $\{\mathbf{x}_0, \mathbf{x}_1\}$ by flipping a coin b, sample $\mathbf{r}$ uniformly at random, and return $\mathbf{c} = \mathbf{Enc}_{\mathcal{PK}}(\mathbf{x}_b, \mathbf{r})$ to the adversary. The adversary does the following (algorithm $\mathcal{A}(\mathbf{c})$) to estimate b:

i. Randomly choose a block B'.

ii. Try to decrypt $\mathbf{c}$ by enumerating the unknown bits of information for every possible dependency tree.

iii. If in any decryption B' is a substring of $\mathbf{x}_i$ and $|B'| > \ell_B/2$, mark $\mathbf{x}_i$ as "possible", $i \in \{0,1\}$.

iv. Upon finishing the enumeration, if exactly one among $\mathbf{x}_0$ and $\mathbf{x}_1$ is *possible* then output its index.

v. Otherwise answer by flipping an unbiased coin.

Because the enumeration step only enumerates $\mathrm{O}(\log N)$ bits, it is easy to verify that $\mathcal{A}$ has running time $\mathsf{poly}(N)$. Now, let us calculate the success advantage of the adversary.

Consider the case $\mathbf{c} \in \mathbf{Enc}_{\mathcal{PK}}(\mathbf{x}_0)$. The block B' hits the maximal output block

[3]We use the following notational convention. When the random choices r of **Enc** are not denoted, by **Enc** we refer to the set of outputs of **Enc** one for each possible r and a fixed input x.

B with probability at least $1/N$. Since $\ell_B(\mathbf{c}) \leq N$,

$$\Pr_{\mathbf{x}_0 \sim \mathcal{U}_N, \mathbf{Enc}} \left[\ell_B(\mathbf{c}) > \frac{1}{2}\ell_B\right] = \Pr_{\mathbf{c}} \left[\ell_B(\mathbf{c}) > \frac{1}{2}\ell_B\right] > \frac{\ell_B}{2N}$$

Notice that when $B' = B$ and $\ell_B(\mathbf{c}) > \ell_B/2$, $\mathbf{x}_0$ is always marked "possible". Therefore,

$$\begin{aligned}
&\Pr_{\mathbf{x}_0 \sim \mathcal{U}_N, \mathbf{Enc}} \left[\mathbf{x}_0 \text{ is "possible"} \,\middle|\, \mathbf{c} \in \mathbf{Enc}_{\mathcal{PK}}(\mathbf{x}_0)\right] \\
> &\Pr_{\mathbf{c}} \left[\ell_B(\mathbf{c}) > \frac{1}{2}\ell_B \,\middle|\, B' = B\right] \Pr_{B'}[B' = B] > \frac{\ell_B}{2N^2}
\end{aligned}$$

Then, we calculate that false positive probability $\Pr_{\mathbf{x}_1 \sim \mathcal{U}_N, \mathbf{c}}[\mathbf{x}_1 \text{ is "possible"} \,|\, \mathbf{c} \in \mathbf{Enc}_{\mathcal{PK}}(\mathbf{x}_0)]$. For every $\mathbf{c}$, there are at most $2^{\mathrm{O}(\log N)} = N^{\mathrm{O}(1)}$ decrypted B' with sufficient length, each of which appears in randomly chosen $\mathbf{x}_1$ with probability $< N/2^{\ell_B/2}$. Thus

$$\begin{aligned}
&\Pr_{\mathbf{x}_1 \sim \mathcal{U}_N, \mathbf{c}}[\mathbf{x}_1 \text{ is "possible"} \,|\, \mathbf{c} \in \mathbf{Enc}_{\mathcal{PK}}(\mathbf{x}_0)] \\
= &\sum_{\mathbf{c} \in \mathbf{Enc}_{\mathcal{PK}}(\mathbf{x}_0)} \Pr[\mathbf{c}] \Pr_{\mathbf{x}_1 \sim \mathcal{U}_N}[\mathbf{x}_1 \text{ is "possible"} \,|\, \mathbf{c}] \\
\leq &\sum_{\mathbf{c} \in \mathbf{Enc}_{\mathcal{PK}}(\mathbf{x}_0)} \Pr[\mathbf{c}] \sum_{B' \text{ decrypted from } \mathbf{c}} \Pr_{\mathbf{x}_1 \sim \mathcal{U}_N}[B' \text{ is a substring of } \mathbf{x}_1 \text{ and } |B'| > \frac{\ell_B}{2}] \\
< &\sum_{\mathbf{c} \in \mathbf{Enc}_{\mathcal{PK}}(\mathbf{x}_0)} \Pr[\mathbf{c}] \sum_{B' \text{ decrypted from } \mathbf{c}} \frac{N}{2^{\ell_B/2}} \leq N^{\mathrm{O}(1)} \cdot \frac{N}{2^{\ell_B/2}} \\
= &\frac{N^{\mathrm{O}(1)}}{2^{\ell_B/2}} = \frac{N^{\mathrm{O}(1)}}{2^{\omega(\log N)}} = N^{\mathrm{O}(1)-\omega(1)} = N^{-\omega(1)}
\end{aligned}$$

Combining above two inequalities, when $\mathbf{x}_0 \leftarrow \mathcal{U}_N$, $\mathbf{r} \leftarrow \mathcal{U}_{|\mathbf{r}|}$, and $\mathbf{c} = \mathbf{Enc}_{\mathcal{PK}}(\mathbf{x}_0, \mathbf{r}) \in \mathbf{Enc}_{\mathcal{PK}}(\mathbf{x}_0)$, the adversary succeeds with advantage

$$\Pr_{\mathbf{r} \leftarrow \mathcal{U}_{|\mathbf{r}|}}[\mathcal{A}(\mathbf{c}) = 0] - \Pr_{\mathbf{x}_1 \leftarrow \mathcal{U}_N, \mathbf{r} \leftarrow \mathcal{U}_{|\mathbf{r}|}}[\mathcal{A}(\mathbf{c}) = 1] > \frac{\ell_B}{2N^2} - N^{-\omega(1)}$$

Because the case $\mathbf{c} \in \mathbf{Enc}_{\mathcal{PK}}(\mathbf{x}_1)$ is symmetric, we have

$$\Pr[\mathcal{A} \text{ reports } b \text{ “possible” }] - \Pr[\mathcal{A} \text{ reports } (1-b) \text{ “possible” }] > \frac{\ell_B}{2N^2} - N^{-\omega(1)}$$

Therefore, $\mathcal{A}$ succeeds with probability significantly larger than $\frac{1}{2}$,

$$\Pr_{\mathbf{c}}[\mathcal{A}(\mathbf{c}) = b] > \frac{1}{2} + \frac{1}{2}(\frac{\omega(\log N)}{N^2} - N^{-\omega(1)}) = \frac{1}{2} + \frac{1}{N^{\mathrm{O}(1)}}$$

However, such an adversary $\mathcal{A}$ implies that the public-key system is not IND-CPA secure. We conclude that for any IND-CPA secure public-key system with a streaming computable decryption scheme, the private key must have length $\Omega(N)$. □

4.3 Lower Bounds for Randomness Extractors

We lower bound the number of passes for all general (Section 4.3.1) and oblivious (Section 4.3.2) streaming extractors. Note that both lower bounds are obtained with nemesis bit-fixing sources. Moreover, in the proof we allow free access to the random seed (even if the seed length d is greater than the local memory size), which makes the lower bound even stronger.

4.3.1 Lower Bound for General Extractors

Consider the streaming algorithm that computes an extractor Ext by making in total p passes over constant many streams and using local memory of size s. If the algorithm makes $\Omega(\log n)$ passes, then we already have $p = \Omega\left(\log\log\frac{1}{\varepsilon}\right)$ for every $\varepsilon = 1/2^{\mathsf{poly}(n)}$, e.g. $\varepsilon = 2^{-n^2}$, which is sufficiently small for any meaningful discussion[4]. Also, the streaming algorithm is barely interesting if the output length is close to the local memory size, e.g. $m \leq \mathrm{o}(s \log n)$.

We obtain the following lower bound for the case $p = \mathrm{o}(\log n)$ and $m \geq s \log n$.

[4]For randomness extractors, n denotes the length of each sample from the weak source. The input length of an extractor is usually $n + \mathsf{polylog}(n)$, but certainly no more than $\mathsf{poly}(n)$, even when the extractor uses multiple sources.

Theorem 4.9. *Fix arbitrary positive integer $\lambda \geq 1$ and an error tolerance $\varepsilon > 0$. Suppose the extractor $\mathsf{Ext} : (\{0,1\}^n)^\lambda \times \{0,1\}^d \to \{0,1\}^m$ outputs ε-close to uniform strings on input of λ-many bit-fixing sources each of min-entropy k together with a d-bit long seed, and Ext is computable by a (p,s,t)-streaming algorithm. If t is a constant, $n - k \geq \Omega(n)$, $k > m = s \cdot (\log \frac{1}{\varepsilon})^{\Omega(1)}$ and $d \leq \mathrm{O}(s)$, then the streaming algorithm must make at least $p = \Omega\left(\log\log \frac{1}{\varepsilon}\right)$ passes.*

Proof. To construct a nemesis weak source we first introduce the parameters specifying the dependency graph $\Gamma(x^{(1)}, \ldots, x^{(\lambda)}, y)$ induced by the computation process of $\mathsf{Ext}(x^{(1)}, \ldots, x^{(\lambda)}, y)$, where recall that the $x^{(i)}$'s come from the weak sources and y is the random seed. $\Gamma(x^{(1)}, \ldots, x^{(\lambda)}, y)$ is uniquely specified by $\mathsf{Ext}(x^{(1)}, \ldots, x^{(\lambda)}, y)$ since the computation of Ext is deterministic. Let $\ell \stackrel{\text{def}}{=} t^p = n^{\mathrm{o}(1)}$ upper bound the number of leaf nodes in every dependency tree in $\Gamma(x^{(1)}, \ldots, x^{(\lambda)}, y)$, where each leaf node is an input block. We partition the input (length λn) equally into $b \stackrel{\text{def}}{=} \left\lceil \frac{\lambda \ell n}{n - k - \log(1+\lambda) - t^p(p+1)(p\log t + \log n)} \right\rceil = \mathrm{O}(\lambda\ell) = \mathrm{o}(n)$ many blocks each of size $\lambda n/b$, where for simplicity from this point on we assume that $b|n$. Then, there are at most $b_O \stackrel{\text{def}}{=} t^p b$ output blocks by Corollary 4.4 (page 100). Furthermore, the number g of different choices of dependency tree is upper bounded as follows.

$$g \leq \prod_{i=1}^{p+1} (bt^{i-1})^{t^{p+1-i}} < (b_O)^{l(p+1)} = (t^p b)^{t^p(p+1)}$$

Now, we define a function $F : \{0,1\}^{\lambda n} \times \{0,1\}^d \to [g]$ such that for every $x^{(1)}, \ldots, x^{(\lambda)} \in \{0,1\}^n, y \in \{0,1\}^d$, $F(x^{(1)}, \ldots, x^{(\lambda)}, y) = z$, where z is the distinct index of the dependency tree rooted at the largest[5] output block A in the dependency graph $\Gamma(x^{(1)}, \ldots, x^{(\lambda)}, y)$. Such a function F is well-defined since $\Gamma(x^{(1)}, \ldots, x^{(\lambda)}, y)$ is unique for fixed $(x^{(1)}, \ldots, x^{(\lambda)}, y)$. Moreover, since A is the largest among the b_O many output blocks, it contains at least m/b_O output bits.

By averaging there are at least $2^{\lambda n + d}/g$ many tuples $(x^{(1)}, \ldots, x^{(\lambda)}, y)$ sharing the same output z_0 under F. Then, by a simple probabilistic argument we conclude that there exist sets $S'_1, \cdots, S'_\lambda \subseteq \{0,1\}^n$, $|S'_i| \geq \frac{2^n}{(1+\lambda)g}$ such that for every $x =$

[5] If there are more than one block being largest then we can choose i as the index of any of them.

$\left(x^{(1)},\ldots,x^{(\lambda)}\right) \in S' \stackrel{\text{def}}{=} S'_1 \times \cdots \times S'_\lambda$, there exists $R_x \subseteq \{0,1\}^d$ satisfying $|R_x| \geq \frac{2^d}{(1+\lambda)g}$ and for all $y \in R_x, F(x^{(1)},\ldots,x^{(\lambda)},y) = z_0$. For convenience, we denote by $\mathsf{Tree}(z_0)$ the dependency tree indexed by z_0 and note that $|S'| = \prod_{i=1}^{\lambda} |S'_i| \geq 2^{\lambda n}/((1+\lambda)g)^\lambda$.

Now, we lower bound the number of $x \in S'$ inducing identical content in all the leaf nodes $\mathsf{Tree}(z_0)$. Recall that every dependency tree has at most ℓ leaf nodes (input blocks), in particular at most $\ell \cdot \frac{\lambda n}{b} = \lambda \ell n/b$ input bits are contained in the leaf nodes of $\mathsf{Tree}(z_0)$. Moreover, their indices are fully determined by z_0. Let D_{z_0} be the set consisting of those indices, then $|D_{z_0}| \leq \lambda \ell n/b$ and $x|_{D_{z_0}}$ contains exactly the content of all leaf nodes of $\mathsf{Tree}(z_0)$ on input x. Partition $D_{z_0} = \left(D^{(1)}_{z_0},\ldots,D^{(\lambda)}_{z_0}\right)$ such that $D^{(i)}_{z_0}$ is the part of D_{z_0} consistent with $x^{(i)}$.

For every i, by averaging there is $x^{(i)}_0 \in \{0,1\}^{|D^{(i)}_{z_0}|}$ and a sufficiently large subset $S''_i \subseteq S'_i$, such that $|S''_i| \geq |S'_i|/2^{|D^{(i)}_{z_0}|}$, and moreover if $x \in S'' \stackrel{\text{def}}{=} S''_1 \times \cdots \times S''_\lambda$, then the values of the bits in leaf nodes of $\mathsf{Tree}(z_0)$ is fixed to $x_0 = \left(x^{(1)}_0,\ldots,x^{(\lambda)}_0\right)$. Formally, $x|_{D_{z_0}} = x_0$ for every $x \in S''$, and

$$|S''| \geq \prod_{i=1}^{\lambda} \frac{|S'_i|}{2^{|D^{(i)}_{z_0}|}} \geq \frac{|S'|}{2^{|D_{z_0}|}} \geq \frac{|S'|}{2^{\lambda\ell n/b}} \tag{4.6}$$

Recalling that for every $x \in S'$ there exists $R_x \subseteq \{0,1\}^d, |R_x| \geq \frac{2^d}{(1+\lambda)g}$, such that for every $y \in R_x$, the dependency tree rooted at the largest output block A is $\mathsf{Tree}(z_0)$. If furthermore $x \in S''$, then $\mathsf{Tree}(z_0)$ has only $x|_{D_{z_0}} = x_0$ in its leaf nodes. That is, for every $x \in S''$ and every $y \in R_x$, the largest output block A acquires all its information entropy from the local memory during the computation, since the input blocks it depends on have fixed content. Note that there are no more than t^p intermediate nodes in $\mathsf{Tree}(z_0)$, while to each node at most s bits entropy can be transferred from local memory. Thus, for such x and y, A acquires at most $t^p s$ bits information entropy during the entire computation.

Following the specific choice of b on page 109, we have

$$b = \left\lceil \frac{\lambda\ell n}{n - k - \log(1+\lambda) - t^p(p+1)(p\log t + \log n)} \right\rceil \geq \frac{\lambda\ell n}{n - k - \log(1+\lambda) - \log g}$$

Therefore, $|S''_i| \geq \frac{|S'_i|}{2^{\lambda\ell n/b}} \geq \frac{2^n}{(1+\lambda)g \cdot 2^{\lambda\ell n/b}} \geq 2^k$.

Now, we are ready to present the nemesis bit-fixing sources. We denote by $\mathcal{U}_{S_i}$ the uniform distribution over $S_i \stackrel{\text{def}}{=} \left\{ x^{(i)} \in \{0,1\}^n \middle| \, x^{(i)}|_{D_{z_0}^{(i)}} = x_0^{(i)} \right\}$. Note that $\mathcal{U}_{S_i}$ is bit-fixing by definition, and since $2^k \leq |S_i''| \leq |S_i| = 2^{n-|D_{z_0}^{(i)}|}$, $\mathcal{U}_{S_i}$ is also an (n,k)-source. Let $S = S_1 \times \cdots \times S_\lambda$, then $\mathcal{U}_S$ is the corresponding product distribution, and moreover, $|S| = \prod_{i=1}^{\lambda} |S_i| = \prod_{i=1}^{\lambda} 2^{n-|D_{z_0}^{(i)}|} = 2^{\lambda n - |D_{z_0}|}$.

Finally, for this specific nemesis bit-fixing source, we lower bound the statistical distance between the extraction output $\mathsf{Ext}(\mathcal{U}_S, \mathcal{U}_d)$ and the uniform distribution $\mathcal{U}_m$.

Fix arbitrary $z_0 \in [g]$. Denote by $\mathcal{E} = \mathcal{E}(x,y)$ the event $F(x,y) = z_0$. Then, for every $x \in S'' \subseteq S$, we have $R_x \subseteq \{0,1\}^d$ such that $|R_x| \geq \frac{2^d}{(1+\lambda)g}$ and for every $y \in R_x$ there is $F(x,y) = z_0$ and hence $\mathcal{E}(x,y)$. Recall that $|S''| \geq |S'|/2^{|D_{z_0}|}$ by (4.6), $|S| = 2^{\lambda n - |D_{z_0}|}$, and $|S'| \geq \frac{2^{\lambda n}}{(1+\lambda)^\lambda g^\lambda}$,

$$
\begin{aligned}
&\Pr_{x \leftarrow \mathcal{U}_S, y \leftarrow \mathcal{U}_d}[\mathcal{E}(x,y)] \\
&\geq \Pr_{x \leftarrow \mathcal{U}_S}[x \in S''] \cdot \Pr_{y \leftarrow \mathcal{U}_d}[\mathcal{E}(x,y) \;\big|\; x \in S''] \\
&\geq \frac{|S''|}{|S|} \cdot \min_{x \in S''} \Pr_{y \leftarrow \mathcal{U}_d}[y \in R_x] \\
&\geq \frac{|S'|/2^{|D_{z_0}|}}{2^{\lambda n - |D_{z_0}|}} \cdot \min_{x \in S''} \frac{|R_x|}{2^d} \\
&\geq \frac{|S'|}{2^{\lambda n}} \cdot \frac{1}{(1+\lambda)g} = \frac{1}{(1+\lambda)^{1+\lambda} g^{1+\lambda}}
\end{aligned}
$$

On the other hand, the possible outputs of A conditioned on $\mathcal{E}$ are at most $2^{t^p s}$, since A is fully determined by the $\leq t^p s$ bits from the local memory. By conditioning on $\mathcal{E}$ and taking into account the position of A in the output, there are $2^{t^p s} m$ distinct choices for A while there are at most $2^{m - m/b_O}$ possibilities for the rest $m - m/b_O$ bits in the output. Thus, there are at most $2^{t^p s} m \cdot 2^{m - m/b_O}$ possible outputs conditioned on $\mathcal{E}$ (by union bound). Let $Z \subseteq \{0,1\}^m$ be defined as follows.

$$
Z \stackrel{\text{def}}{=} \left\{ z \in \{0,1\}^m \;\middle|\; \exists x \in \{0,1\}^{\lambda n}, \exists y \in \{0,1\}^d, \text{such that } F(x,y) = z_0, \mathsf{Ext}(x,y) = z \right\}
$$

It immediately follows that $|Z| \leq 2^{t^p s} m \cdot 2^{m - m/b_O}$, and in fact $Z = \left\{ \mathsf{Ext}(x,y) \;\middle|\; \mathcal{E}(x,y) \right\}$ is the range of Ext given that $\mathcal{E}(x,y)$ holds.

Therefore, we lower bound the statistical distance between $\mathsf{Ext}(\mathcal{U}_S, \mathcal{U}_d)$ and $\mathcal{U}_m$ by

$$\begin{aligned}
\mathcal{SD}\left(\mathsf{Ext}(\mathcal{U}_S, \mathcal{U}_d), \mathcal{U}_m\right)) &= \frac{1}{2} \sum_{z \in \{0,1\}^m} \left|\Pr[\mathsf{Ext}(\mathcal{U}_S, \mathcal{U}_d) = z] - \Pr[\mathcal{U}_m = z]\right| \\
&\geq \frac{1}{2} \sum_{z \in Z} \left(\Pr[\mathsf{Ext}(\mathcal{U}_S, \mathcal{U}_d) = z] - \frac{1}{2^m} \right) \\
&= \frac{1}{2} \left(\Pr[\mathsf{Ext}(\mathcal{U}_S, \mathcal{U}_d) \in Z] - \frac{|Z|}{2^m} \right) \\
&\geq \frac{1}{2} \left(\Pr_{x \leftarrow \mathcal{X}, y \leftarrow \mathcal{U}_d} [\mathcal{E}(x, y)] - \frac{|Z|}{2^m} \right) \\
&\geq \frac{1}{2(1+\lambda)^{1+\lambda} g^{1+\lambda}} - \frac{2^{t^p s} m \cdot 2^{m - m/b_O}}{2^{m+1}} \\
&\geq \frac{1}{2(1+\lambda)^{1+\lambda} (t^p b)^{t^p (p+1)(1+\lambda)}} - 2^{t^p s + \log m - 1 - m/b_O}
\end{aligned}$$

Since Ext is a (k, ε)-extractor, $\varepsilon \geq \mathcal{SD}\left(\mathsf{Ext}(\mathcal{U}_S, \mathcal{U}_d), \mathcal{U}_m\right))$ by definition. Thus, when $m/b_O = \omega(t^p s + t^p (p+1)(p \log t + \log b))$

$$\varepsilon \geq \Omega\left(\frac{1}{(t^p b)^{2t^p (p+1)(1+\lambda)}} \right)$$

In conclusion, we get $p = \Omega\left(\log\log(\frac{1}{\varepsilon})\right)$ for every $m = s \cdot (\log \frac{1}{\varepsilon})^{\Omega(1)}$. □

4.3.2 Lower Bound for Oblivious Extractors

A streaming algorithm is *oblivious* if there is a predetermined sequence according to which the head moves on streams.

Definition 4.10. *A streaming extractor* Ext *is an* oblivious streaming extractor *if after fixing the input length and the random seed, its head moves on the streams depending only the time step. In other words, for fixed random seed* y, $\mathsf{Ext}_y(\cdot) \stackrel{def}{=} \mathsf{Ext}(\cdot, y)$ *is an oblivious streaming algorithm,*

For oblivious extractors, we obtain the following lower bound, which is exponentially separated from the oblivious RRB extractor which makes $\log\log(1/\varepsilon)$-passes.

Theorem 4.11. *Suppose* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *is a* (k, ε)*-extractor computable obliviously with a* (p, s, t)*-streaming algorithm where* $p = \mathrm{o}(\log n)$ *and* t *is a*

constant. Then, for every constants $\alpha, \beta \in (0,1)$ *and for every* $\varepsilon < \frac{1}{2}, k = n^{1-\alpha}, m \geq k^{1-\beta}$*, and* $d \leq \frac{m}{2t^{2p}} = m^{1-\mathrm{o}(1)}$*, there must be* $s \geq m^{\Omega(1)} = n^{\Omega(1)}$*. Moreover, this holds for bit-fixing sources.*

This lower bound indicates that constructing a streaming extractor for sources of min-entropy $o(n)$ with sub-logarithmic number of passes is far from what we know today. It is hard to imagine how an extractor can intelligently adapt its computation (i.e. head moves) based on the specific content of the given sample. To the best of our knowledge, every other extractor in the literature is oblivious at the time of the writing of this dissertation, except from von Neumann's extractor [von51] (which works only under very strong independence assumptions).

The following theorem implies Theorem 4.11.

Theorem 4.12. *Suppose* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *is a* (k,ε)*-extractor for bit-fixing sources and it is computable obliviously with a* (p,s,t)*-streaming algorithm where* $p = \mathrm{o}(\log n)$ *and* t *is a constant. Then, for every* $k = \mathrm{o}\left(\frac{n}{t^p}\right)$ *and constant* $\varepsilon < \frac{1}{2}$*, and for every* $b > \frac{t^p}{1-2\varepsilon-kt^p/n} = \mathrm{O}(t^p)$*, there must be* $s > \frac{m-t^pbd-t^pb}{t^{2p}b}$*.*

In particular, for $m = n^{\Omega(1)}$ and $d \leq m^{0.99}$, this theorem implies that $s \geq m^{1-\mathrm{o}(1)}$.

Proof. First, we consider the dependency graph $\Gamma(x,y)$ with the input equally partitioned into b blocks, where each block has length n/b. By Corollary 4.4 (page 100), there are at most t^pb output blocks and hence the longest output block, denoted by A, contains at least $m/(t^pb)$ bits. Then, there are $c = t^p$ many leaf nodes in the dependency tree T_A rooted at A, since it is a t-branching tree with depth p.

Going over all 2^d choices of random seeds, every specific input block would be hit $2^dc/b$ times in average one input block appears as a leaf node in T_A for $\leq 2^dc/b$ distinct random seeds (since a single seed may induce a T_A hitting the same input block multiple times). Thus, there exists $\left\lceil \frac{kb}{n} \right\rceil$ input blocks that are hit by at most $\left(\left\lceil \frac{kb}{n} \right\rceil \frac{c}{b} \right) \cdot 2^d$ distinct random seeds.

Suppose the input is a bit-fixing source X with all its entropy concentrated in those $\left\lceil \frac{kb}{n} \right\rceil$ input blocks. Then, only $\leq \left(\left\lceil \frac{kb}{n} \right\rceil \frac{c}{b} \right) \cdot 2^d$ distinct random seeds are able

to induce T_A with at least one unfixed leaf node. That is, with probability $1-\left\lceil\frac{kb}{n}\right\rceil\frac{c}{b}$, all the leaf nodes in T_A are fixed. In such case, A is fully determined by the local memory when the heads enter any intermediate nodes in T_A, which is bounded by $t^p s$ since there are less than t^p intermediate nodes in T_A. Thus, A is fully determined by the $t^p s$ bits from local memory and r bits from the seed. For this $1-\left\lceil\frac{kb}{n}\right\rceil\frac{c}{b}$ fraction of random seeds, each possible output in A appears with probability at least $2^{-(d+t^p s)}$.

Finally, we lower bound the statistical distance between $\mathsf{Ext}(X,\mathcal{U}_d)$ and uniform distribution with their restriction on the output block A,

$$\begin{aligned}
&\mathcal{SD}\left(\mathsf{Ext}(X,\mathcal{U}_d),\mathcal{U}_m\right)\\
\geq&\mathcal{SD}\left(\mathsf{Ext}(X,\mathcal{U}_d)|_A,\mathcal{U}_{|A|}\right)\\
\geq&\left(1-\left\lceil\frac{kb}{n}\right\rceil\frac{c}{b}\right)\cdot 2^d\cdot 2^{t^p s}\left(\frac{1}{2^{d+t^p s}}-\frac{1}{2^{|A|}}\right)\\
\geq&\left(1-\left\lceil\frac{kb}{n}\right\rceil\frac{c}{b}\right)\cdot 2^d\cdot 2^{t^p s}\left(\frac{1}{2^{d+t^p s}}-\frac{1}{2^{m/(t^p b)}}\right)\\
\geq&\left(1-\left\lceil\frac{kb}{n}\right\rceil\frac{c}{b}\right)\left(1-\frac{1}{2^{m/(t^p b)-d-t^p s}}\right)
\end{aligned}\tag{4.7}$$

Assume for contradiction that $s\leq\frac{m-t^p bd-t^p b}{t^{2p}b}$. Then, $m/(t^p b)-d-t^p s\geq 1$ and hence

$$\frac{1}{2^{m/(t^p b)-d-t^p s}}\leq\frac{1}{2}$$

Plugging it into (4.7), and recalling that $c=t^p$ (the number of leaf nodes in T_A, defined on page 113),

$$\mathcal{SD}\left(\mathsf{Ext}(X,\mathcal{U}_d),\mathcal{U}_m\right)\geq\left(1-\left\lceil\frac{kb}{n}\right\rceil\frac{c}{b}\right)\cdot\frac{1}{2}=\frac{1}{2}-\left\lceil\frac{kb}{n}\right\rceil\frac{t^p}{2b}\geq\frac{1}{2}-\frac{kt^p}{2n}-\frac{t^p}{2b}$$

Therefore, $\mathcal{SD}\left(\mathsf{Ext}(X,\mathcal{U}_r),\mathcal{U}_m\right)>\varepsilon$ for every constant $\varepsilon<\frac{1}{2}$, $k=\mathrm{o}\left(\frac{n}{t^p}\right)$, $p=\mathrm{o}\left(\log n\right)$, and $b>\frac{t^p}{1-2\varepsilon-kt^p/n}=\mathrm{O}\left(t^p\right)$. This contradicts the condition that Ext is a (k,ε)-extractor.

Therefore, it must be the case that $s>\frac{m-t^p br-t^p b}{t^{2p}b}$. □

4.3.3 Remarks and Future Directions

The lower bounds in Section 4.3.1 and Section 4.3.2 complement the RRB construction in Section 3.1. They are tight in number of passes and in min-entropy rate for bit-fixing and affine sources. In other words, $\Omega(\log\log n)$ many passes and min-entropy $k = \Omega(n)$ are sufficient and necessary for $\varepsilon = 1/\mathsf{poly}(n)$ and for affine sources.

The only gap is that RRB is proved (Section 3.2.4 on page 78) to extract randomness from $\mathrm{O}(\log n)$ many independent samples for general (n,k)-sources, whereas the ideal extractor only takes one sample.[6] In the practical randomness extraction method (Section 3.3 on page 82) that works on real-world data, we use RRB as a single-source extractor and this approach (including the extractor and other components) is empirically validated by our experimental results (see Section 3.4).

We conjecture that RRB works on a single sample from an arbitrary adversary (n,k)-source as long as $k = \Omega(n)$. It remains of theoretical interest to prove this conjecture, although such "arbitrary adversary (n,k)-sources" do not exits in practice.

Conjecture 4.13. *For every $\varepsilon > 0$ and $k = \Omega(n)$, there is $b = \Theta(\log^2 \frac{n}{\varepsilon}), d = \mathrm{O}(b \log n), \ell = \Omega\left(n/\log^2 \frac{n}{\varepsilon}\right)$, and $m = \Omega\left(n/\log^2 \frac{n}{\varepsilon}\right)$, such that for every (n,k)-source $\mathsf{RRB} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k,ε)-extractor.*

Another future direction is to study intelligent constructions of non-oblivious randomness extractors, which may open a new page of randomness extraction.

[6] Actually, we have obtained the proof for the general case, i.e. RRB extracts $\Omega(n)$ bits from any single $(n, \Omega(n))$-source, after the dissertation was sent out for review. This result will appear in future publications.

Bibliography

[AB09] S. Arora and B. Barak. *Computational complexity: a modern approach*, volume 1. Cambridge University Press Cambridge, 2009.

[AIK06a] B. Applebaum, Y. Ishai, and E. Kushilevitz. Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162, 2006. (also CCC'05).

[AIK06b] B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in NC^0. *SIAM Journal of Computing (SICOMP)*, 36(4):845–888, 2006. (also FOCS '04).

[AIK08] B. Applebaum, Y. Ishai, and E. Kushilevitz. On pseudorandom generators with linear stretch in NC^0. *Computational Complexity*, 17(1):38–69, 2008. (also RANDOM'06).

[AIK09] B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography with constant input locality. *Journal of Cryptology*, 22(4):429–469, 2009. (also CRYPTO'07).

[AIK10] B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography by cellular automata or how fast can complexity emerge in nature? In *ICS*, pages 1–19, 2010.

[Ale03] Michael Alekhnovich. More on average case vs approximation complexity. In *Foundations of Computer Science (FOCS)*, pages 298–307. IEEE, 2003.

[AM99] L. Arge and P. B. Miltersen. External memory algorithms. chapter On Showing Lower Bounds for External-memory Computational Geometry Problems, pages 139–159. American Mathematical Society, Boston, MA, USA, 1999.

[Ban14] M. Bansal. Big data: Creating the power to move heaven and earth. *MIT Technology Review*, 2014.

[Bar86] D. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38(1):150–164, 1989 (also STOC '86).

[BBD+02] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 1–16, New York, NY, USA, 2002. ACM.

[BH12] P. Beame and T. Huynh. The value of multiple read/write streams for approximating frequency moments. *ACM Transactions on Computation Theory*, 3(2):6, 2012. (also FOCS '08).

[BIW06] B. Barak, R. Impagliazzo, and A. Wigderson. Extracting randomness using few independent sources. *SIAM Journal on Computing*, 36(4):1095–1118, 2006.

[BJP11] J. Bronson, A. Juma, and P. A. Papakonstantinou. Limits on the stretch of non-adaptive constructions of pseudo-random generators. In *Theory of Cryptography Conference (TCC)*, pages 504–521, 2011.

[BJR07] P. Beame, T. S. Jayram, and A. Rudra. Lower bounds for randomized read/write stream algorithms. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 689–698, New York, NY, USA, 2007. ACM.

[BYGW99] Z. Bar-Yossef, O. Goldreich, and A. Wigderson. Deterministic amplification of space-bounded probabilistic algorithms. In *Conference on Computational Complexity (CCC)*, pages 188–198. IEEE, 1999.

[BYRST02] Z. Bar-Yossef, O. Reingold, R. Shaltiel, and L. Trevisan. Streaming computation of combinatorial objects. In *Annual IEEE Conference on Computational Complexity (CCC)*, volume 17, 2002.

[CG88] B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on Computing*, 17(2):230–261, 1988.

[CMBH02] I. J Cox, M. L Miller, J. A Bloom, and C. Honsinger. *Digital watermarking*, volume 53. Springer, 2002.

[CMV13] K-M Chung, M. Mitzenmacher, and S. P. Vadhan. Why simple hash functions work: Exploiting the entropy in a data stream. *Theory of Computing*, 9(30):897–945, 2013.

[CY91] J. Chen and C.-K. Yap. Reversal complexity. *SIAM Journal on Computing*, 20(4):622–638, 1991.

[DEOR04] Y. Dodis, A. Elbaz, R. Oliveira, and R. Raz. Improved randomness extraction from two independent sources. In *RANDOM*, pages 334–344. Springer, 2004.

[die08] The marsaglia random number CDROM including the Diehard Battery of Tests of Randomness. 2008.

[FGF07] Fast Galois Field Arithmetic Library in C/C++. 2007. `http://web.eecs.utk.edu/~plank/plank/papers/CS-07-593/`.

[GHS09] M. Grohe, A. Hernich, and N. Schweikardt. Lower bounds for processing data with few random accesses to external memory. *Journal of the ACM*, 56(3):Art. 12, 58, 2009.

[GMP14] The GNU Multiple Precision Arithmetic Library. 2014.

[Gol01] O. Goldreich. *Foundations of cryptography*. Cambridge University Press, Cambridge, 2001. Basic tools (Vol. I).

[GRS06] A. Gabizon, R. Raz, and R. Shaltiel. Deterministic extractors for bit-fixing sources by obtaining an independent seed. *SIAM Journal on Computing*, 36(4):1072–1094, 2006.

[GS05] M. Grohe and N. Schweikardt. Lower bounds for sorting with few random accesses to external memory. In *Symposium on Principles of Database Systems (PODS)*, pages 238–249, 2005.

[HG01] N. Howgrave-Graham. Approximate integer common divisors. In *Cryptography and Lattices*, pages 51–66. Springer, 2001.

[HILL99] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal of Computing (SICOMP)*, 28(4):1364–1396, 1999. (also STOC'89).

[HRV10] I. Haitner, O. Reingold, and S. Vadhan. Efficiency improvements in constructing pseudorandom generators from one-way functions. In *Symposium on Theory Of Computing (STOC)*, pages 437–446, 2010.

[HS08] A. Hernich and N. Schweikardt. Reversal complexity revisited. *Theoretical Computer Science*, 401(1-3):191–205, 2008.

[ILL89] R. Impagliazzo, L. A. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *Symposium on Theory Of Computing (STOC)*, pages 12–24, 1989.

[IZ89] R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *Foundations of Computer Science (FOCS)*, pages 248–253. IEEE, 1989.

[KGY89] M. Kharitonov, A. V. Goldberg, and M. Yung. Lower bounds for pseudorandom number generators. In *Foundations of Computer Science (FOCS)*, pages 242–247, 1989.

[KZ06] J. Kamp and D. Zuckerman. Deterministic extractors for bit-fixing sources and exposure-resilient cryptography. *SIAM Journal on Computing*, 36(5):1231–1247, 2006.

[MP12] D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. *EUROCRYPT 2012*, pages 700–718, 2012.

[Mut03] S. Muthukrishnan. Data streams: Algorithms and applications. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 413–413, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[MVV14] D. G. Marangon, G. Vallone, and P. Villoresi. Random bits, true and unbiased, from atmospheric turbulence. *Scientific Reports*, 2014.

[NZ96] N. Nisan and D. Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996.

[PAM+10] S. Pironio, A. Acín, S. Massar, A B de La Giroday, Dzimitry N Matsukevich, P Maunz, S Olmschenk, D Hayes, L Luo, T A Manning, et al. Random numbers certified by Bell's theorem. *Nature*, 464(7291):1021–1024, 2010.

[Pap94] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, 1994.

[Pap10] P. A. Papakonstantinou. Constructions, lower bounds, and new directions in cryptography and computational complexity. Ph.D. Thesis, 2010.

[PM13] S. Pironio and S. Massar. Security of practical private randomness generation. *Phys. Rev. A*, 87:012336, 2013.

[PY12] P. A. Papakonstantinou and G. Yang. A remark on one-wayness versus pseudorandomness. In *18th Annual International Computing and Combinatorics Conference (COCOON)*, pages 482–494, 2012.

[PY14] P. A. Papakonstantinou and G. Yang. Cryptography with streaming algorithms. In *CRYPTO (2)*, pages 55–70, 2014.

[Rao09a] A. Rao. Extractors for a constant number of polynomially small min-entropy independent sources. *SIAM J. Comput.*, 39(1):168–194, May 2009.

[Rao09b] A. Rao. Extractors for low-weight affine sources. In *Conference on Computational Complexity (CCC)*, pages 95–101. IEEE, 2009.

[Raz05] R. Raz. Extractors with weak random seeds. In *Symposium on Theory Of Computing (STOC)*, pages 11–20. ACM, 2005.

[Reg05] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Symposium on Theory Of Computing (STOC)*, pages 84–93. ACM, 2005.

[RRV99] R. Raz, O. Reingold, and S. Vadhan. Extracting all the randomness and reducing the error in trevisan's extractors. In *Symposium on Theory Of Computing (STOC)*, pages 149–158. ACM, 1999.

[RSN+10] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Special publication 800–22 (revision 1a), National Institute of Standards and Technology, 2010.

[SB00] J. Soto and L. Bassham. Randomness testing of the advanced encryption standard finalist candidates. Technical report, NIST (NISTIR) 6483, 2000.

[Sha04] R. Shaltiel. *Current trends in theoretical computer science. The Challenge of the New Century. (book chapter)*, volume Vol 1: Algorithms and Complexity. World Scientific, 2004.

[Sha11] R. Shaltiel. An introduction to randomness extractors. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 21–41. Springer, 2011.

[SU01] R. Shaltiel and C. Umans. Simple extractors for all min-entropies and a new pseudo-random generator. In *Foundations of Computer Science (FOCS)*, pages 648–657. IEEE, 2001.

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

[TPI13] The Templated Portable I/O Environment. 2013. `http://madalgo.au.dk/tpie/`.

[Tre99] L. Trevisan. Construction of extractors using pseudo-random generators. In *Symposium on Theory Of Computing (STOC)*, pages 141–148. ACM, 1999.

[UZZ+13] M. Um, X. Zhang, J. Zhang, Y. Wang, S. Yangchao, D-L Deng, L.-M. Duan, and K. Kim. Experimental certification of random numbers via quantum contextuality. *Scientific reports*, 3, 2013.

[Vad04] S. P. Vadhan. Constructing locally computable extractors and cryptosystems in the bounded-storage model. *Journal of Cryptology*, 17(1):43–77, 2004.

[Vit01] Jeffrey Scott Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Comput. Surv.*, 33(2):209–271, June 2001.

[von51] J. von Neumann. Various techniques in connection with random digits. *Applied Math Series*, 12:36–38, 1951.

[VV12] U. V. Vazirani and T. Vidick. Certifiable quantum dice - or, testable exponential randomness expansion. *Philosophical Transactions-Royal Society of London Series A*, 370:3432–3448, 2012.

[VZ12] S. P. Vadhan and C. J. Zheng. Characterizing pseudoentropy and simplifying pseudorandom generator constructions. In *Symposium on Theory Of Computing (STOC)*, pages 817–836, 2012.

[WK13] J. Wakefield and P. Kerley. How "big data" is changing lives. *BBC (News Technology)*, 2013. `http://www.bbc.com/news/technology-21535739`.

[Yao79] A. Yao. Some complexity questions related to distributive computing(preliminary report). In *Symposium on Theory Of Computing (STOC)*, STOC '79, pages 209–213. ACM, 1979.

[ZL77] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.

[Zuc90] D. Zuckerman. General weak random sources. In *Foundations of Computer Science (FOCS)*, pages 534–543. IEEE, IEEE, 1990.

[Zuc96] D. Zuckerman. Simulating BPP using a general weak random source. *Algorithmica*, 16(4-5):367–391, 1996.