On the Use of Randomness Extractors for Practical Committee Selection

by

Zehui Zheng
B.Eng., Shenzhen University, China, 2017

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

On the Use of Randomness Extractors
for Practical Committee Selection

by

Zehui Zheng
B.Eng., Shenzhen University, China, 2017

Supervisory Committee

Dr. Valerie King, Co-supervisor
(Department of Computer Science)

Dr. Jianping Pan, Co-supervisor
(Department of Computer Science)

ABSTRACT

In this thesis, we look into the problem of forming and maintaining good committees that can represent a distributed network. The solution to this problem can be used as a sub-routine for Byzantine Agreement that only costs sub-quadratic message complexity. Most importantly, we make *no* cryptographic assumptions such as the Random Oracle assumption and the existence of private channels. However, we do assume the network to be peer-to-peer, where a message receiver knows who the message sender is. Under the synchronous full information model, our solution is to utilize an approximating disperser for selecting a good next committee with high probability, repeatedly. We consider several existing theoretical constructions (randomized and deterministic) for approximating dispersers, and examine the practical applicability of them, while improving constants for some constructions. This algorithm is robust against a *semi-adaptive* adversary who can decide the set of nodes to corrupt periodically. Thus, a new committee should be selected before the current committee gets corrupted. We also prove some constructions that do not work practically for our scenario.

Contents

List of Tables

List of Figures

List of Symbols

A	A next-bit predictor function
$\mathcal{A}$	A family of next-bit predictor functions
B	The set of bad nodes in the network
b	The number of bins
c	A constant
D	The degree of each left-hand side node in an approximating disperser (or the size of a committee)
d	The number of bits needed to specify a neighbour in disperser (or the number of bits as seed for extractor)
d_e	The degree of a regular expander
E	An extractor that outputs bits that are close to perfect randomness, given a weak random source
$\mathbb{E}$	The expected value of a random variable
f	A function
$\mathbb{F}$	A finite field
G	A graph
K	The number of bad committees from the left-hand side of an approximating disperser
k	The number of input symbols for Reed-Solomon codes
l	The size of a set in a family of (weak) design
M	The size of the network
m	The number of bits needed to specify a node in the network

N	The size of the left-hand side of an approximating disperser (or the number of committee choices)
$\mathbb{N}$	The set of natural numbers
n	The number of bits as the input to an approximating disperser (or extractor)
$\mathcal{O}$	A Random Oracle
P	A predicate (or Boolean function)
q	The size of a finite field
S	A set
$\mathscr{S}$	A combinatorial (weak) design
U	A uniform distribution
V	The set of left-hand side nodes in a bipartite graph
W	The set of right-hand side nodes in a bipartite graph
X	A random variable
Y	A distribution which is also a δ-source
β	The ratio of Byzantine (or bad) nodes in the network
β_c	The ratio of Byzantine (or bad) nodes in the committee
β_n	The ratio of non-random bits in an n-bit string
Δ	The amount of deviation used in the Chernoff bound
δ	The minimum ratio of entropy in a distribution
ϵ	An error parameter to indicate the distance between a distribution from a uniform distribution
Γ	The set of neighbours of a given node
λ_2	The second largest absolute eigenvalue of the transition matrix of the regular expander graph, where transition matrix is equal to the graph's adjacency matrix with each element divided by the degree
μ	The expected value of a random variable
ρ	A parameter related with the overlap between any two sets in a (weak) design
σ	A parameter used in the definition of error correcting codes

ACKNOWLEDGEMENTS

I am very grateful for my supervisor **Valerie King**, who gave me this problem to work on and guided me through the whole problem solving process. It is such a suitable topic that I can learn more about theory while respecting my experience in systems and applications.

I am also deeply thankful for my supervisor **Jianping Pan**'s supportive feedback and constructive suggestions from a system expert's perspectives. I really appreciate Prof. Pan's support for me pursuing where my passion belongs.

Many thanks to Dr. **Hung Le** for countless in-depth discussions for the problems that I faced during the research process and guiding me like my two wonderful supervisors do. Hung generously pointed out my methodologically incorrect strategies in reading theory papers and patiently illustrated how to do research in theory step by step and to understand a research problem in the best way.

Last but not least, thanks to Prof. **Jared Saia**, Prof. **Lin Cai** and our group members for helpful discussions and beneficial feedbacks.

DEDICATION

This thesis is dedicated to all the beautifully important ones who appeared in my Master's life.

Chapter 1

Introduction and Overview

1.1 Introduction

The Bitcoin blockchain [37] is an ingenious way of reaching consensus among a distributed group of nodes over the network. The main idea is to require miners to solve a one-way computational puzzle (proof of work, PoW) for publishing the next block of transactions. Eventually, only the longest chain will survive. Essentially, we can interpret PoW as a scheme to randomly, with probability proportional to the computational power, *elect a leader* from the network, who will propose the next block. If we define *good* nodes as those who follow the given protocol, then so long as good nodes possess more than 50% computational power, over the long term, the valid chain will eventually outpace invalid chains.

Similarly, for Ethereum's chain-based proof of stake (PoS) [49, 8], a leader is elected randomly, with probability proportional to the size of the stake (e.g., the number of coins possessed, etc.), for the next block. So long as good nodes possess more than 50% wealth in the system, with high probability (w.h.p.)[1], dishonest chains will eventually be discarded.

The above two are the pioneers that inspire numerous newly emerged consensus algorithms/protocols being applied into cryptocurrency systems. Besides them, we also have a body of more than forty years of research on the *Byzantine Generals Problem* [31], which is to find algorithms to ensure that good parties will reach agreement. We also call this type of algorithm *Byzantine Agreement* (BA). Although those algorithms typically do not require puzzle solving, without cryptographic assumptions,

[1] With probability of error that goes to 0 polynomially quickly as the input size increases.

agreement can only be achieved when there are less than 1/3 of Byzantine (bad) nodes in the system, who may exhibit arbitrary behaviours.

In comparison with PoW and PoS, BA reaches consensus mainly by exchanging messages, which means that we do not need to burn a large amount of electricity, or build a cryptocurrency system. In order for our algorithms to be lightweight, we will avoid PoW and PoS. However, BA may have a high message complexity. Please see Section 1.4 for detailed related works on message complexity. Moreover, we will not address the Sybil attack, which can occur in permissionless blockchain systems, while PoW and PoS can possibly handle the attack.

A sub-quadratic message complexity can be achieved by forming a "good" committee of size, say D, to represent the whole network. Here, "good" means the committee has more than a required majority threshold ratio of good nodes. Our main idea of forming good committees will rely heavily on the method used by King et al. [29]. King et al. worked against a non-adaptive adversary, while we assume an adversary who slowly takes over nodes. Thus, we will have to select the next committee before the adversary corrupts the current committee. Improved algorithms by King and Saia [28] can work against an adaptive adversary. However, the existence of private channels between all pairs of processors was assumed in [28].

Although we build on the algorithms by King et al., we focus more on the practical implementable constructions of the algorithms. Once we have a practical construction for forming good committees, it can be used as a sub-routine for BA to achieve sub-quadratic message complexity (see King et al. [29]). Further, it can also be integrated into blockchain protocols. For example, Luu et al. [33] designed a scalable blockchain protocol called *ELASTICO*, by sharding the network into smaller committees, each of which in parallel processes a disjoint set of transactions. However, *ELASTICO* does make the Random Oracle assumption. Our goal will be to do away with this assumption.

A Random Oracle $\mathcal{O}$ is defined as a map from $\{0,1\}^*$ to $\{0,1\}^\infty$[2] chosen by selecting each bit of $\mathcal{O}(x)$ uniformly and independently, for every x, according to Bellare et al. [6]. However, it is pointed out by Canetti et al. [9] that a security proof in the Random Oracle Model does not mean that there are no "structural flaws" in the scheme when we replace the Random Oracle with a real world implementation, such as a cryptographic hashing function.

[2]Notation: $\{0,1\}^*$ and $\{0,1\}^\infty$ denote the space of finite and infinite binary strings, respectively. ∞ is used instead of specifying how long "sufficiently long" is [6].

Therefore, the *motivation* of this thesis work is to look into the problem of finding a practically implementable algorithm of forming good committees, while making *no* cryptographic assumptions, except that a message receiver knows who the message sender is. We make no limitation on adversary's computational power and the adversary can see every node's state.

We use existing typical BA protocols which may have quadratic message complexity as a primitive to achieve BA among the selected good committee of size D. Our message complexity is $O(D^2) + O(D(M - D)) = O(DM)$ for each *binary* BA to be reached for the network. BA among the committee contributes the term $O(D^2)$, and then spreading the agreement to the rest of the nodes in the network contributes the $O(D(M - D))$ term.

The main focus of this thesis becomes how we form a good committee. The naive solution would be to select D nodes from the network uniformly at random. By concentration bounds [35], we can guarantee that with probability $1 - e^{-\Omega(D)}$, the selected committee is good. However, in a distributed system, where there is no central node to trust, there is no easy way to agree on perfect randomness, not even for a single bit.

Approximating dispersers (Zuckerman [51]), or equivalently[3] randomness extractors and samplers, can help to mitigate this predicament, in the sense that even if the source is not perfectly random, the output would be a committee that has a ratio of bad members over the committee size not too far away from the expected bad ratio (β). The definitions for those objects will be discussed in the later technical chapters. Theoretically, there exist various methods for constructing approximating dispersers, either deterministic or randomized (see survey [19, 38, 46]). We will look into the practical aspects of those construction methods, although some of them may not be easy to figure out all the constants.

1.2 Problem Statement

In this section, we clearly state the network model that we use, the problem that we are addressing and the assumptions that we make.

The network model that we use is the *synchronous, peer-to-peer, full information* model.

[3] We may use these terms "approximating dispersers", "randomness extractors" and "samplers" back and forth to refer the same kind of objects, where the variance among them is not significant.

By *synchronous*, we mean all link delays are bounded and the network is synchronized by rounds, where a message sent in round i must be received before round $i+1$ [40]. More precisely, each processor keeps a local clock to keep track of rounds. One can also think of the entire system as if it is driven by one global clock. The machine cycle of each processor can be described as composed of the following three steps: 1. Send messages to (some of) the neighbours; 2. Receive messages from (some of) the neighbours; 3. Perform some local computation. The consumed time for local computation is assumed to be negligible, compared to message transmission time.

By *peer-to-peer*, we mean that each processor can directly send messages to any other processor, without passing intermediary entities [45]. Even when we use the word "broadcast", we still mean that the sender sends message(s) to other nodes one by one, where the sender may not send the same version of the message to all receivers and the message may not be sent to all nodes in the network. A receiver always knows the identity of the sender.

In the *full information model*, we assume the adversary has unbounded computational power and every node has full knowledge of state of the network [23]. This implies that when every node contributes bits, the adversary can first observe what bits good nodes have generated before deciding its own bits.

In this thesis, we build our BA protocol on top of existing typical BA protocols, and we use them as a black box. The problem that this thesis focuses on is to improve the BA protocols by forming committees that represent the whole network, with the help of randomness extractors (or specifically approximating dispersers). We denote BA_D as the primitive that we call to reach BA among the given D nodes. An *epoch* is the time period starting from a committee is selected to be in charge until the next committee is selected. We further assume that the first committee of the system is a good committee, as bootstrapping for the system. Starting from there, the system mainly works as follows:

1. Before the next epoch, if there is a (binary) decision to make, the current committee members call BA_D to reach consensus among the committee. Once a consensus is reached, every committee member will send the consensus to all other nodes who are not in the current committee. Since all nodes know who are in the current committee, they will trust the result sent by the majority (or any specified threshold) of committee members.

2. To select the next committee, the current committee will again call BA_D to

agree on an input string for the approximating disperser. Once nodes in the network learn about the input string, they can efficiently figure out who are in the next committee and the next epoch will commence.

The following property is a direct consequence from the above described system scheme.

Every node in the network always knows who are in the committee, so long as the system is running.

We argue this is true by simple induction.

The base case is that every node in the network knows who are in the first committee. This is straightforward as the bootstrapping assumption suggests.

As for the inductive step, assume that every node in the network knows who are in the current committee (epoch i). Then by epoch $i+1$, the current committee will agree on an input string for the approximating disperser, from which every node in the network will figure out who are in the next committee after learning the agreed string from the current committee.

Since both the base case and the inductive step hold, it holds that every node in the network always knows who are in the committee.

Thus, the problem is *reduced* to how to select the next committee, given the current committee is good, while guaranteeing with high probability that the next committee is also good. This is the main problem that this thesis is addressing. Also, it is worth mentioning that the reasons that we let committees take turn include:

- To ensure (somewhat) fairness, in the sense that every node in the network has a non-zero chance to be in the committee, although not all nodes have the exact same chance.

- To ensure robustness. As the adversary will only have information about the current committee and, from their messages, possibly the very next one, it will not have too much time to use this information to selectively corrupt nodes to its advantage.

We claim that we work with a *semi-adaptive* adversarial model. We give the definition of *semi-adaptive* adversary and comparisons with adaptive and non-adaptive ones [27] as follows:

- Adaptive adversary: an adaptive adversary can take over nodes at any time during the protocol, as long as the number of Byzantine nodes is no more than the tolerable number.
- Non-adaptive adversary: a non-adaptive adversary is one that chooses the set of Byzantine nodes at the beginning of the protocol.
- Semi-adaptive adversary: a semi-adaptive adversary can actively decide the set of nodes to corrupt, under the condition that the time needed by the adversary to corrupt a node is longer than 2 epochs and the number of Byzantine nodes is no more than the tolerable number.

We consider the Byzantine nodes during any epoch to be the set of nodes which are corrupted or become corrupted any time during the epoch. We assume the ratio of the size of this set over the network size is bounded by β. Because we assume the adversary requires at least two epochs to corrupt a node, this set must be decided by the adversary at least before the start of the previous epoch.

What rate of corruption we can tolerate from the adversary depends on the time needed to select the next committee. Specifically, it is determined by the time needed to agree on an input string for the approximating disperser. If we assume each committee member contributes one bit, then it is the time needed to do *reliable broadcast*, which has one additional round than BA, plus one more round for committee members to send out the string to non-committee nodes. We take BA as a primitive, and for round complexity, please see Section 1.4 for more details.

To summarize, we address the problem of selecting the next good committee, assuming:

1. Time is synchronized;
2. Network is peer-to-peer;
3. The adversary has full knowledge of the network, namely no secrecy;
4. The first committee is good (for bootstrapping);
5. We assume a semi-adaptive adversary who can corrupt other nodes as long as the ratio of Byzantine nodes over the network is below β, but these corruptions cannot be fulfilled in a period of time shorter than 2 epochs;

6. Every node in the network knows everyone else's ID;

7. When receiving a message, receiver knows the ID of the sender;

8. Communication links are reliable, which means that there is no packet loss during transmission.

The target that we aim for is summarized as Table 1.1 shows. The numbers are based on a private conversation with a cryptocurrency researcher *Prateek Saxena* [33], and also our knowledge and experience. Ideally, the time used for local computation is ignored.

We think a network of size $1,000,000$ should be reasonable, where we target to form committees of sizes $1,000$. Take the P2P system *OblivP2P* [21] as an example, the throughput is 3.29 MBps for network of size 2^{21}, which is roughly 2 million. If we consider the time needed for selecting the next committee as 10 mins, then we require each committee node to be able to handle up to 2 GB messages. Typically, commercial devices have 128 GB storage space. And we can only tolerate 1/3 of network nodes being Byzantine, without using cryptography. Moreover, we believe failure probability of 2^{-40} can be negligible.

Table 1.1: Parameters that we aim for

Parameters	Values
Network size (M)	$1,000,000$
Committee size (D)	$1,000$
Number of rounds each epoch	$10 \sim 20$
Size of messages received per committee node per epoch	2 GB
Required storage space per node	128 GB
Byzantine ratio over the network (β)	1/3
Failure probability	2^{-40}

1.3 Contributions

1. We create a tool for executing repeated BA for a large network with sub-quadratic message complexity and can be used in blockchain protocols, while making no cryptographic assumptions, such as the Random Oracle assumption and private channels, except that a message receiver knows who the message sender is.

2. We show an example of a random approximating disperser and how it can be used for committee selection, although it has a description too large to be stored or distributed as a whole.

3. We improve the constants for the construction of extractors by Raz et al. [42]. For details, please see Theorem 3.3.1. We have a small part of the construction that is random, so that we can have a construction that has good performance while still can be described compactly.

4. We prove that if the description for random walks on expanders only comes from the current committee, then there is no guarantee that the next committee will stay as good.

5. To the best of our knowledge, this is the first work on examining the construction of extractors to be used in a distributed network.

1.4 Related Works

Historically, BA required a high message complexity. In 1985, it was proved by Dolev and Reischuk [10] that any deterministic protocol in a synchronous model requires $\Omega(M^2)$ messages, where M is the number of nodes in the network. Thus, to achieve sub-quadratic message complexity, we have to resort to randomized solutions. A breakthrough was made by King et al. [29, 30], who designed a scalable leader election and further BA protocol that the number of bits each good node sends and processes is only polylogarithmic in M. The brief idea is to build a layered network, in which each node is a committee. On layer 0, processors[4] are assigned to nodes using a bipartite graph, which is termed as an averaging sampler later on, and similarly for upper layers. To elect subcommittees for higher layers all the way to elect a leader for the top layer, authors adapted the lightest bin algorithm by Feige [11]. Agreement is reached for all but a $O(1/\log M)$ fraction of good processors with high probability, against a non-adaptive adversary who can send an unbounded number of messages. This agreement result can also be termed as "almost-everywhere agreement". Moreover, "everywhere agreement" can be reached from "almost-everywhere agreement" by King et al. [26] and in a load-balanced fashion by King et al. [25], with message

[4]In order to not confuse the audience between the terms of "nodes in the peer-to-peer network" and "nodes in the layered network", when illustrating the related works by King et al., we use the term "processors" to mean "nodes in the peer-to-peer network".

complexity $\widetilde{O}(M^{1.5})$. Further breakthrough was made by King and Saia [28] to achieve $\widetilde{O}(M^{1.5})$ message complexity against an adaptive adversary, assuming the existence of private channels between all pairs of processors. Braud-Santoni et al. [7] proposed a new almost-everywhere to everywhere algorithm with amortized message complexity $\widetilde{O}(1)$ per node. This algorithm can be composed with the almost-everywhere agreement algorithm by King et al. [29, 30] to yield a BA that only has a polylogarithmic message complexity, against a non-adaptive adversary. Lately, Robinson et al. [44] proposed "almost-everywhere" algorithms that have $\widetilde{O}(M)$ message complexity, against a *late* adaptive adversary who has full knowledge of the network state at the beginning of the *previous* round.

Now we look into the round (time) complexity for BA. It was proved in [12] that the lower bound on the number of rounds for deterministic BA algorithms is $t + 1$, where t is the number of Byzantine nodes. Thus, to get around this lower bound, we again need to resort to randomized algorithms. As illustrated above, by achieving "almost-everywhere agreement" and then "from almost-everywhere to everywhere agreement", King et al. [29, 30, 26, 25] proposed algorithms that have time complexity $\widetilde{O}(1)$, against non-adaptive adversary. Again, Braud-Santoni et al. [7] composed with "almost-everywhere agreement" by King et al. also achieves time complexity $\widetilde{O}(1)$. Much better round complexity can be achieved if cryptographic assumptions are used. Note that we only use BA as a primitive, which does not necessarily use cryptographic assumptions. With the assumption of private channels, Katz and Koo [24] showed a BA protocol that takes expected 23 rounds, against a computationally unbounded adversary and less than 1/3 of all nodes being Byzantine nodes. In Abraham et al. [1], it assumed public key cryptography and a random leader oracle, and reached BA in expected 10 rounds against less than 1/2 of all nodes being Byzantine nodes.

For more detailed comparisons of message and time complexity among various models, please refer to a 2010 *SIGACT News* article by King and Saia [27].

For constructions of approximating dispersers, Zuckerman [51] showed that they are essentially equivalent to constructions of randomness extractors. See surveys on constructions of extractors or approximating dispersers [19, 38, 46], which covered a number of ways for constructions: 1. Leftover Hash Lemma, via (almost) universal hash functions; 2. Pseudorandom generators with random predicates; 3. Random walks on an expander; 4. Converting a source to a block-wise source and then extracting randomness from the block-wise source; 5. Condensing a low entropy source to a high entropy source and then extracting randomness from the high entropy source.

It is also possible that some constructions have a composition of several above methods. A related line of research is done by David Zuckerman et al. [39, 47, 51, 52]. Guruswami et al. [16] used list-decodable codes of Parvaresh and Vardy to construct near-optimal randomness condensers and further randomness extractors that are optimal up to constant factors.

1.5 Thesis Outline

The rest of the thesis is organized as follows.

In Chapter 2, we look into the randomized construction for approximating dispersers and formally illustrate our algorithm for selecting the next committee. A specific example of applying a randomly constructed approximating disperser and the drawback of randomized construction will also be shown in this chapter.

Chapter 3 presents a PRG[5]-based construction for randomness extractors, and then an extractor can be converted to a corresponding approximating disperser. In this chapter we improve some constants for the construction. We also have a small part of the construction randomized and numerically show the randomized performance.

We discuss another construction method in Chapter 4, namely random walks on an expander. However, we will show how it cannot give any guarantee on the next committee staying as good as the current committee, if the description of random walks comes from the current committee.

Chapter 5 concludes this thesis and discusses about future works.

[5]PRG: pseudorandom generator.

Chapter 2

Approximating Dispersers and Our Algorithm

In this chapter, we will formally define an approximating disperser and other essential concepts, following which we illustrate how approximating dispersers are closely related with our problem, namely how to select the next committee. We first show how approximating dispersers can be utilized in our problem conceptually, and then we show what is the best possible we can do with them and design our committee forming algorithm accordingly.

2.1 Preliminaries

In this section, we give definitions and preliminaries that we may need throughout the thesis. We mostly follow the definitions from Zuckerman [51]. However, we made modifications for the ease of reading and consistency of symbols.

Definition 2.1.1. *An (N, M, D, K, ϵ)-approximating disperser is a bipartite multigraph on independent sets V and W, such that $|V| = N = 2^n$, $|W| = M = 2^m$, the degree of every node in V is $D = 2^d$, and the following holds. Let $f : W \to [0, 1]$ be arbitrary, and call $v \in V$ bad if $\left| \left(\frac{1}{D} \sum_{w \in \Gamma(v)} f(w) \right) - \mathbb{E}\, f \right| > \epsilon$. We denote $\Gamma(v)$ as the neighbors of v and $\mathbb{E}\, f := \sum_{w \in W} \frac{f(w)}{|W|}$. Then the number of bad vertices in V is at most K.*

Remark 2.1.1. *Approximating dispersers and other kinds of dispersers [52] are related but they are different kinds of objects.*

Definition 2.1.2. *(Zuckerman [52]) The min-entropy of a distribution X is $H_\infty(X) = \min_{x \in X}\{-\log_2 \Pr[X = x]\}$.*

Definition 2.1.3. *A distribution Y on $\{0,1\}^n$ is called a δ-source if for all $x \in \{0,1\}^n$, $Y(x) \leq 2^{-\delta n}$. Note that we use the notation $Y(x)$ to denote the probability of x occurs according to distribution Y.*

Remark 2.1.2. *We also call δ the min-entropy ratio for the n-bit string source Y with min-entropy δn. Min-entropy can be interpreted as the number of truly random bits.*

Definition 2.1.4. *Let Y_1 and Y_2 be two distributions on the same space S, within which X is a set. We use the notation $Y_1(X)$ to denote the probability that every element of set X occurs according to distribution Y_1, and similarly for $Y_2(X)$. The variation distance (or statistical distance) between Y_1 and Y_2 is*

$$||Y_1 - Y_2|| = \max_{X \subseteq S} |Y_1(X) - Y_2(X)| = \frac{1}{2} \sum_{s \in S} |Y_1(s) - Y_2(s)|$$

Definition 2.1.5. *$EXT : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is an $(n, m, d, \delta, \epsilon)$-extractor if, for x chosen according to any δ-source on $\{0,1\}^n$ and y chosen uniformly at random from $\{0,1\}^d$, $EXT(x, y)$ is within statistical distance ϵ from the uniform distribution on $\{0,1\}^m$.*

Definition 2.1.6. *We define* good *nodes as those which follow the given protocol (algorithm). Conversely, we define* Byzantine (bad) *nodes as those who could exhibit arbitrary behaviours.*

Definition 2.1.7. *We define a committee as* good *if (generally) more than $2/3$ of its members are good, unless this ratio is specified. On the contrary, a* bad *committee is one which is not good.*

2.2 Conceptual Usefulness

Approximating dispersers are highly useful for the following reasons:

First, for any given n-bit string as input, an approximating disperser outputs D m-bit strings. If we take each m-bit string as an ID for a node in the network, then the output space of an approximating disperser can cover a network of size up to 2^m. And

thus, each n-bit string specifies a committee of size D. We can view this mapping as a bipartite graph with the set of left-hand side nodes as the set of committee choices (V) and the set of right-hand side nodes as the set of nodes in the network (W), where $|V| = N$, $|W| = M$ and left degree is D. This bipartite view is illustrated in Fig. 2.1.

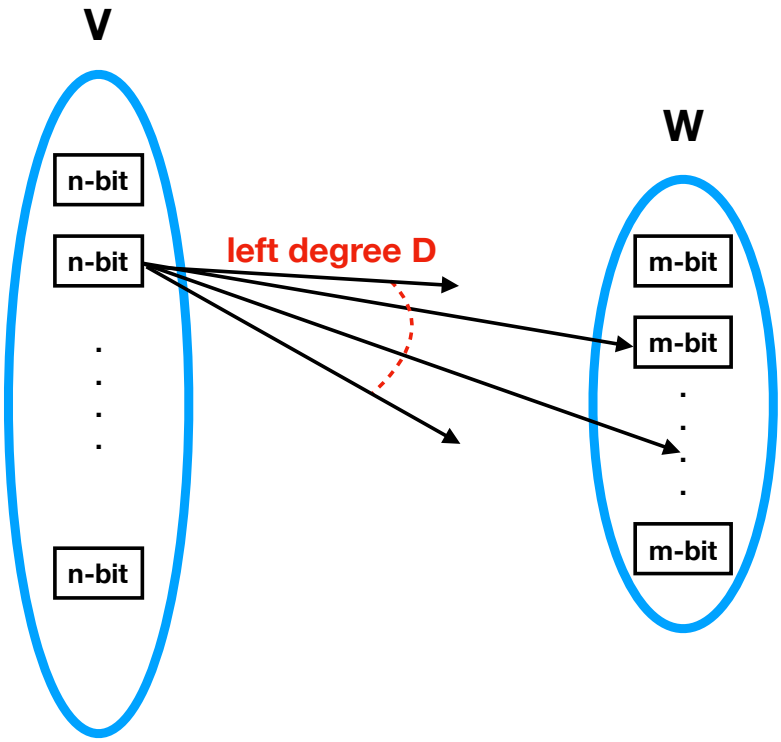

Figure 2.1: The bipartite view of an approximating disperser.

Second, given any set of nodes S[1] in the network, the number of committees that contain much more (or much fewer) nodes from S than the expected value is bounded by K. To state this formally (refer to Definition 2.1.1), we define an arbitrary function f that takes input as an m-bit string and outputs 1 if the input m-bit is the ID for a Byzantine node, and 0 otherwise. Thus, the expectation of such a function is the ratio of Byzantine nodes over the whole network, which we define as $\beta := \mathbb{E}\, f$. We further denote the ratio of Byzantine nodes in any committee as β_c. Then we call a committee *bad* if $|\beta_c - \beta| > \epsilon$. According to the definition for an (N, M, D, K, ϵ)-approximating disperser, the number of bad committee choices is bounded by K.

[1]We can interpret S as the set of Byzantine nodes. S can be any set as long as the ratio of the number of Byzantine nodes over the network is less than β and S is decided at least before the start of the previous epoch.

In our application, we will set the value of ϵ as we aim for committees having ratio of Byzantine nodes to all nodes no more than $\beta + \epsilon$. For BA with no cryptographic assumption, we know that we need to have $\beta + \epsilon < 1/3$ in order to reach BA. The set of committees that have $\beta_c - \beta > \epsilon$ is a subset of the set of committees that have $|\beta_c - \beta| > \epsilon$, which in turn has size bounded by K. Therefore, if we assume that we have uniform random n-bit strings, then the probability that we select a bad committee is no more than $\frac{K}{N}$. However, it is not straightforward to have an agreed upon n-bit string that is uniform random. Instead, the input n-bit string contains some bits that are not uniform random, which are set by an adversary. We define the ratio of the number of bits that are not uniform random over n as β_n. The source that provides such n-bit strings is called $(1-\beta_n)$-source as defined in Definition 2.1.3. We also call this as a bit-fixing source [22], as some bits can be fixed while others are random. With similar analysis as in Zuckerman [52], without a perfect random source, the probability that we select a bad committee will grow by a factor of at most $2^{\beta_n n}$. This is because in the worst case, the adversary will first observe what good nodes' bits are, before deciding its $\beta_n n$ controlled bits. That enables the adversary to try up to $2^{\beta_n n}$ possibilities.

We thus have the following corollary.

Corollary 2.2.1. *Given an (N, M, D, K, ϵ)-approximating disperser and an input n-bit string from a $(1-\beta_n)$-source, where $\epsilon < 1/3 - \beta$, the probability that an approximating disperser outputs a bad committee (size: D) is bounded by $2^{\beta_n n}\frac{K}{N}$, which can also be written as $\frac{K}{N^{1-\beta_n}}$.*

2.3 Randomized Construction

In the previous section, we illustrate why approximating dispersers are useful and how they can be potentially applied towards solving the problem of selecting the next committee. In this section, we will look at what are the best possible properties we can achieve to construct one such object. Specifically, we will give a randomized construction for approximating dispersers.

Proposition 2.3.1. *[Proposition 2.7 [51]] If there is an efficient $(n, m, d, \delta, \epsilon)$-extractor, then there is an efficiently constructible $(2^n, 2^m, 2^d, 2^{1+\delta n}, \epsilon)$-approximating disperser.*

Proposition 2.3.2. *[Proposition 2.19 [51]] let positive n, m, δ and ϵ be given, and set $k = \delta n$. Suppose $D \geq (2\ln 2)(2^{m-k} + n - k + 3)/\epsilon^2$. Then there is an $(n, m, d =$*

$\log D, \delta, \epsilon)$-*extractor. Note that we can take* $k = m$*, in which case* $D = O(n/\epsilon^2)$ *and* $d = \log n + 2\log \epsilon^{-1} + O(1)$.

Corollary 2.3.1. *With failure probability at most* 2^{1-M}*, a randomly constructed bipartite graph with* N *left vertices,* M *right vertices and left degree* $D \geq (2\ln 2)(\log N - \log K + 5)/\epsilon^2$*, is an* (N, M, D, K, ϵ)*-approximating disperser, where* $K = 2M = 2^{1+k}$.

This corollary can be obtained from Proposition 2.3.1 and Proposition 2.3.2 by taking $k = m$. For each vertex on the left-hand side of the bipartite graph, we pick D nodes uniformly at random. It is shown in the proof for Proposition 2.3.2 [51] that with this construction, we obtain an $(n, m, d, \delta, \epsilon)$-extractor with failure probability 2^{1-M}. Lastly, Proposition 2.3.2 gives us a corresponding $(N = 2^n, M = 2^m, D = 2^d, K = 2^{1+\delta n}, \epsilon)$-approximating disperser.

2.4 Committee Selection Algorithms

In this section, assuming there is an existing current committee, we give two algorithms for selecting the next committee. One algorithm is the slightly modified version of the other. We first assume that $D \leq n$ and give the first algorithm. The second algorithm can be applied when $D > n$. Both algorithms are from a node's point of view.

2.4.1 $D \leq n$

In Algorithm 1, each member of the committee contributes ℓ bits, which are concatenated as an n-bit string (add padding if necessary), where $n \geq D \cdot \ell$.

In Algorithm 1, line 7 is done in parallel. We also assume that BA_D is able to handle the case when a Byzantine committee member does not send messages at all, in which case set its bit string to all 0's.

We claim that it is possible to have an approximating disperser with $D \leq n$. The reason is that, for $(n, m, d, \delta, \epsilon)$-extractors with $\delta < 1 - 1/n$ and $\epsilon < 1/2$, Nisan and Zuckerman [39] give a lower bound that $d \geq \max(\log \epsilon^{-1} - 1, \log((1-\delta)n))$, namely $D \geq \max(1/(2\epsilon), (1-\delta)n)$. Thus, it is possible to have $D = (1-\delta)n \leq n$.

However, if we use the randomized construction given in Corollary 2.3.1, then we have following lemma.

Lemma 2.4.1. *With the randomized construction,* $D \leq n$ *only if* $m \geq 62$.

Algorithm 1: Select The Next Committee (when $D \leq n$)

if *not in Current Committee* **then**
  wait until knowing the BA_D consensus from the majority of current committee members;
end
if *in Current Committee* **then**
  generate a uniform random bit string with length ℓ, where $\ell = \lfloor n/D \rfloor$;
  broadcast the bit string to all other committee members;
  call BA_D to agree on every committee member's bit string;
  sort strings according to members' IDs;
  concatenate all D strings;
  if $D \cdot \ell < n$ **then**
    add $(n - D \cdot \ell)$ 0's as padding
  end
  broadcast the concatenated string to all non-committee members;
  set boolean value for being in current committee as false;
end
given input bit string, calculate the set (S) of output strings from the approximating disperser;
set the set of current committee to S

Proof. From Corollary 2.2.1, we know that $\epsilon < 1/3 - \beta$.

$$\begin{aligned} D &\geq (2\ln 2)(\log N - \log K + 5)/\epsilon^2 \\ &\geq (2\ln 2)(\log N - \log 2M + 5)/\epsilon^2 \\ &\geq (18\ln 2)(n - m + 4) \end{aligned} \tag{2.1}$$

For D to be less than or equal to n, the following equation must be satisfied:

$$(18\ln 2)(n - m + 4) \leq n \tag{2.2}$$

From Eq. (2.2), we can get:

$$n \leq \frac{(18\ln 2)m - 72\ln 2}{18\ln 2 - 1} \tag{2.3}$$

As we know, the number of "bad" committee choices is $K = 2M$ out of totally N

choices, namely $K = 2M \leq N$ and $2^{1+m} \leq 2^n$. Thus, we have:

$$\frac{(18 \ln 2)m - 72 \ln 2}{18 \ln 2 - 1} \geq m + 1 \tag{2.4}$$

This implies $m \geq 62$, which is a necessary condition for "$D \leq n$" to be true. And thus, if $m < 62$, then $D > n$.

□

When $D > n$, Algorithm 1 will not work: If every committee member contributes only one bit, there are totally D bits, which is larger than the required number of input bits for the approximating disperser. Thus, we will need to truncate the D-bit string into an n-bit string somehow, which will be discussed in the next subsection.

2.4.2 $D > n$

The naive solution to truncate the D-bit string into an n-bit string is take the first n bits of the D-bit string (bits sorted according to members' IDs). However, this method will fail in the case when the number of bad bits is significant. In this case, if we arbitrarily take the first n bits, those n-bits may compose a very high ratio of non-random bits. Thus, a more robust way of truncating strings is to adopt the lightest bin algorithm used in King et al. [29], which is based on Feige's algorithm [11]. That is, when a committee member contributes a bit, that bit will be followed by a bin choice. Eventually, we use the bits that fall into the lightest bin (the bin that contains the least number of bits) and concatenate them according to bit sender's ID. If the number of bits in the lightest bin is less than n, we add 0's as padding. This algorithm is shown as Algorithm 2.

Again, in Algorithm 2, line 8 is done in parallel. We also assume that BA_D is able to handle the case when a Byzantine committee member does not send messages at all, in which case set its bit string to all 0's. In line 9, when using bits in the bin, we only use one bit from each sender into that bin.

2.5 An Example

In this section, we give a detailed example of applying Algorithm 2 in a network for selecting the next committee and analyze the performance in terms of the probability that we fail to select a good committee.

Algorithm 2: Select The Next Committee (when $D > n$)

if *not in Current Committee* **then**
  wait until knowing the BA_D consensus from the majority of current committee members;
end
if *in Current Committee* **then**
  set the number of bins $b = \lceil D/n \rceil$;
  generate a uniform random bit suffixed with uniform random $\log b$ bits for a bin choice;
  broadcast the bit string to all other committee members;
  call BA_D to agree on every committee member's bit string;
  pick the bin with the least number of bits in it;
  set ℓ to be the number of bits in the lightest bin;
  sort bits according to senders' IDs;
  concatenate all sorted bits;
  if $\ell < n$ **then**
    add $(n - \ell)$ 0's as padding to the end of string;
  end
  broadcast the string to all non-committee members;
  set boolean value for being in current committee as false;
end
given input bit string, calculate the set (S) of output strings from the approximating disperser;
set the set of current committee to S

We first determine our settings in a network of size 2^{20}, which is about one million nodes. Assume that we are given the first committee as a good committee, our goal is to select the next committee that is also good. Lastly, we aim for the size of a committee to be about a thousand.

According to Algorithm 2, we know that the probability that we fail to select a good committee consists of three parts, assuming the probability that BA_D fails is negligible:

1. The probability that we fail to construct an approximating disperser;
2. The failure probability when applying Feige's algorithm. That is, we may not end up with the n-bit string with enough good bits as we target for;
3. Given an n-bit string that has enough good bits, the probability that we still select a bad committee.

The first part is directly given in Corollary 2.3.1, as $2^{1-M} = 2^{1-2^{20}}$ (denote as $\Pr[E1]$) being the probability that we fail to randomly construct an $(N, M = 2^{20}, D, K = 2^{21}, \epsilon)$-approximating disperser.

According to Corollary 2.2.1, we know that the larger the N (or equivalently n), the lower the failure probability for part 3. However, larger n will also result in a larger D, that is the committee size. Here we take an example of setting $N = 2^{132}$, $\epsilon = 0.3$ and $\beta = 1/3 - \epsilon = 1/30$. Thus, $D \geq (2\ln 2)(\log N - \log K + 3)/\epsilon^2 \approx 1787$. We take $D = 2048 = 2^{11}$ to be our committee size.

Assume now we have a good current committee with size 2^{11}, Feige's algorithm can be applied in order to agree on a bit string of length $n = 132$ as an index for the next committee. As there will be $D = 2048$ bits if every committee member contributes one bit, we will need $b = \lceil D/n \rceil = 16$ bins to guarantee the lightest bin always has no more than $n = 132$ bits and 0's will be appended as padding if there are not enough bits in the lightest bin. More specifically, every good current committee member generates a random bit, concatenated with a random bin choice (4 bits), and send out these 5 bits. Current committee will do BA (by calling BA_D repeatedly) to agree on every member's 5-bit string. After an agreement on every member's 5-bit string is done, we only use those random bits (only use one bit from each sender) in the lightest bin. To analyze the quality of the bits in the lightest bin, we bound the probability that the lightest bin contains a good bit ratio that deviates from the expected ratio by a certain amount. To do that, we first apply the *Chernoff bound* to bound the probability that a given bin has deviation more than what we wanted and then apply *union bound* for the probability that any of 16 bins have this large deviation [35].

Define the random variable R as the number of random (good) bits in a bin and μ as the expected value of R. We further define the good bit ratio deviation as Δ. Thus, for any given bin and $0 < \Delta < 1$, we have:

$$\Pr[R \leq (1-\Delta)\mu] \leq \left(\frac{e^{-\Delta}}{(1-\Delta)^{(1-\Delta)}}\right)^{\mu} \tag{2.5}$$

In our settings, a good committee has more than 2/3 good members, thus $\mu > D \cdot \frac{2}{3}/b = 85.33$. If we take $\Delta = 0.55$, then for a given bin:

$$\begin{aligned}\Pr[R \leq (1-0.55)\mu] &= \Pr[R \leq 38.4] \\ &\leq \left(\frac{e^{-0.55}}{(1-0.55)^{(1-0.55)}}\right)^{85.33} \\ &\approx 2^{-23.47}\end{aligned} \tag{2.6}$$

After applying union bound over all $b = 16$ bins, we obtain the probability that any of those bins has $R \leq 38.4$, denoted as event $E2$:

$$\begin{aligned}\Pr[E2] &:= b \cdot \Pr[R \leq 38.4] \\ &\approx 16 \cdot 2^{-23.47} \\ &= 2^{-19.47}\end{aligned} \tag{2.7}$$

Correspondingly, the probability that all bins (including the lightest bin) have at least 39 good bits is $1 - 2^{-19.47}$.

As the last step, assume that we have a 132-bit string that contains at least 39 good bits, which means the number of non-random bits is $\beta_n n = 132 - 39 = 93$. According to Corollary 2.2.1, the probability that, given a $(1-\beta_n)$-source from the current committee, the next committee is bad (event $E3$) is:

$$\begin{aligned}\Pr[E3] &:= 2^{\beta_n n}\frac{K}{N} \\ &= 2^{93}\frac{2^{21}}{2^{132}} \\ &= 2^{-18}\end{aligned} \tag{2.8}$$

Over all, the probability that the current committee fails to select the next good committee is:

$$\begin{aligned}\Pr[Fail] &= 1 - (1 - \Pr[E1])(1 - \Pr[E2])(1 - \Pr[E3]) \\ &= 1 - (1 - 2^{1-2^{20}})(1 - 2^{-19.47})(1 - 2^{-18}) \\ &< 2^{-17.55}\end{aligned} \tag{2.9}$$

We summarize what we have achieved in the above example, compared with what

we aim for, shown in Table 2.1. We have BA_D in the table because we use it as a primitive for BA. Thus, the cost for time and messages can vary, depending on the implementation for BA_D. For more related works on time and message complexity on BA, please see Section 1.4. Number of rounds needed to select the next committee is at least $2 + BA_D$ because there will be one additional round for committee members to send out their choices of bit strings to other committee members and one final round for committee members to send out the consensus to non-committee nodes. The size of messages is approximately $D \cdot BA_D$ because we will do BA in parallel for each member from the committee of size D. The required storage space will be explained in the next section.

Table 2.1: Parameters that we achieved for randomized approximating dispersers

Parameters	Target	Achieved
Network size (M)	$1,000,000$	$1,048,576$
Committee size (D)	$1,000$	2048
Number of rounds each epoch	$10 \sim 20$	$2 + BA_D$
Size of messages received per committee node per epoch	2 GB	$D \cdot BA_D$
Required storage space per node	128 GB	10^{34} GB
Byzantine ratio over the network (β)	$1/3$	$1/30$
Failure probability	2^{-40}	$2^{-17.55}$

2.6 The Catch

For the example given in Section 2.5, the probability that the current committee selects a bad committee is as small as $2^{-17.55}$. However, there is one catch.

Since the approximating disperser is generated randomly at the beginning of the system, we assume this graph is a part of the program to be distributed to every node in the network. But if we take the size of the approximating disperser into consideration, we would find this graph being enormously large to store or distribute. Take the approximating disperser generated in Section 2.5 as an example, it is basically a bipartite graph with 2^{132} nodes on the left-hand side and 2^{20} nodes on the right-hand side, while the left degree is 2^{11}. If we represent this bipartite graph as an adjacency list, without any compression, it will need $ND \log M = 2^{132} \times 2^{11} \times 20$ bits for storage, which is equivalent to 10^{34} GB. This makes it almost impossible to be stored on any commercial devices.

One solution might be to utilize a pseudo random number generator so that we only need to distribute the seeds, whose size is relatively small. However, it is non-trivial to prove that the same construction of approximating dispersers using pseudo randomness can give similar properties as using perfect randomness. We thus leave this as a future work.

Therefore, we will look into other construction methods that might have more succinct descriptions.

Chapter 3

PRG-based Randomness Extractors

In this chapter, we look into another construction method for approximating dispersers that only needs a small part as randomized, and thus the description is small. Moreover, the randomized performance is *checkable*. That means this construction method is provably good.

Due to Proposition 2.3.1, we know that extractors and approximating dispersers are essentially equivalent. Thus, it is an option to construct an extractor first and then convert it to the corresponding approximating disperser.

3.1 Overview

Leftover Hash Lemma (LHL) [17, 20] can be used as a simple deterministic construction for a randomness extractor. More specifically, it states that (almost) universal hash functions are good (strong) randomness extractors. However, it requires a large seed length (as large as n for universal hash functions). Barak et al. [4] revisited LHL and addressed this issue by *Expand-then-Extract* approach, which expands seed length via a pseudorandom generator (PRG). However, the output bits are computationally indistinguishable from uniform random rather than statistically close to it.

Loosely, two distributions are computationally indistinguishable if no efficient algorithms can tell the two distributions apart [14]. Nevertheless, Goldreich and Krawczyk [15] proved the existence of sparse pseudorandom distributions, which are "probability distributions concentrated in a very small set of strings, yet it is infea-

sible for any polynomial-time algorithm to distinguish between truly random coins and coins selected according to these distributions." In other words, there are distributions that are computationally indistinguishable from each other but statistically very different [14]. As we mentioned, our extractors are defined with regard to being statistically close to uniform distributions, according to Definition 2.1.5, while computational indistinguishability is not what we need.

Trevisan [48] demonstrates a surprising connection between extractors and pseudorandom generators and shows that every pseudorandom generator of a certain kind is an extractor. Most importantly, it outputs bits that are statistically close to, rather than computationally indistinguishable from, the uniform distribution. The main idea is to use the Nisan-Wigderson (NW) generator together with an error correcting code. Raz et al. [42] improved the result to further reduce the number of truly random bits needed, by replacing the "combinatorial designs" used in [48] with a weaker notion (weak designs). We follow the definitions and proofs in [42] for the ease of reading. However, some constants are modified as we target a construction which is practical and thus constants play a significant role.

Fig. 3.1 is a high-level view of how we would construct the extractors we need. We take the NW generator as our extractor. NW generator is composed of a weak design and a random predicate that is a string from a δ-source but encoded by the error correcting code. The error correcting code is a concatenated code from a Reed-Solomon code and a Hadamard code. Overall, we only need a weak design and an error correcting code embedded into the program that we assume to be distributed to nodes in the network.

Figure 3.1: The structure of a randomness extractor construction.

3.2 Preliminaries

In this section, we give definitions and preliminary knowledge about (weak) designs, NW generators, and error correcting codes.

Definition 3.2.1. *([42]) For $l \in \mathbb{N}$ and $\rho \geq 1$, a family of sets $S_1, ..., S_{m'} \subset [d]$ is an (l, ρ)-design if*

1. *For all i, $|S_i| = l$.*
2. *For all $i \neq j$, $|S_i \cap S_j| \leq \log \rho$.*

Definition 3.2.2. *([42]) For $l \in \mathbb{N}$ and $\rho \geq 1$, a family of sets $S_1, ..., S_{m'} \subset [d]$ is a weak (l, ρ)-design if*

1. *For all i, $|S_i| = l$.*
2. *For all $i \neq j$, $\sum_{j<i} 2^{|S_i \cap S_j|} \leq \rho(m' - 1)$.*

An (l, ρ)-design implies a weak (l, ρ)-design, since $\sum_{j<i} 2^{|S_i \cap S_j|} \leq \sum_{j<i} 2^{\log \rho} \leq \rho(m' - 1)$.

Definition 3.2.3. *For a string $y \in \{0,1\}^d$, define $y|_S$ as the string of length $|S|$ by selecting the bits indexed by S from y.*

For example, if we take $d = 10$, $y = \{0011001100\}$ from $\{0,1\}^{10}$, and $S = \{1, 3, 10\}$, then $y|_S = \{010\}$. (We let index start from 1.)

Definition 3.2.4. *([42]) Let $\mathscr{S} = (S_1, ..., S_{m'})$ be a collection of subsets of $[d]$ of size l, and let $P : \{0,1\}^l \to \{0,1\}$ be any Boolean function. Then the Nisan-Wigderson generator $NW_{\mathscr{S},P}$ is defined as $NW_{\mathscr{S},P}(y) = P(y|_{S_1})P(y|_{S_2})...P(y|_{S_{m'}})$.*

Note that a Boolean function P, which we also call as a *predicate*, can be described as a string of bits that work as the truth table of the predicate. An example of a truth table is shown in Table 3.1. For this predicate of two-bit inputs, we can encode the truth table as $\{1011\}$. In the following, we may use the *description* of a Boolean function (predicate) to mean the bit string encoding the truth table of that Boolean function (predicate).

Table 3.1: An example of a truth table with description $\{1011\}$

x	$P(x)$
00	1
01	0
10	1
11	1

The purpose of introducing error correcting codes is to increase the minimum (relative) Hamming distance between codewords, plus upper bounding the number of codewords in any Hamming spheres with certain diameter.

Definition 3.2.5. *The Hamming distance between two codewords is the number of bit positions in which they differ. The relative Hamming distance is defined as Hamming distance divided by the length of codewords.*

It is important to note that the relative Hamming distance between the descriptions of two predicates implies one minus the probability that these two predicates output the same bit, given uniform random input. Namely, if we denote RHD as a function that outputs the relative Hamming distance between two predicates $P1$ and $P2$, and x is uniform random from the input space to the predicates, then $\Pr[P1(x) = P2(x)] = 1 - RHD(P1, P2)$. For example, let the description of $P1$ be $\{00110011\}$ and the description of $P2$ be $\{11110011\}$. Then the relative Hamming distance between $P1$ and $P2$ is $1/4$, because their descriptions differ 2 output cases out of totally 8 output cases. If x is uniform random from $\{0,1\}^3$, then $\Pr[P1(x) = P2(x)] = 1 - 1/4 = 3/4$.

Generally speaking, our goal is for any two predicates to have low probability of outputting the same bit. As we will show in the proof in the next section, we want to bound the probability that the output bit of a random predicate can be predicted, which is what a pseudorandom generator tries to avoid. Since this problem can be converted to increase the minimum relative Hamming distance between the descriptions of any two predicates, now it is natural to introduce the definition for error correcting codes.

Definition 3.2.6. *([42]) For $\sigma > 0$, let $EC_{n,\sigma} : \{0,1\}^n \to \{0,1\}^{\bar{n}}$ be any error correcting code whose minimum relative Hamming distance is at least $1/2 - \sigma^2/2$.*

Remark 3.2.1. *([5]) After encoding by $EC_{n,\sigma}$, every Hamming ball of relative radius $1/2 - \sigma$ in $\{0,1\}^{\bar{n}}$ contains at most $1/(3\sigma^2)$ codewords.*

We will use the output of $EC_{n,\sigma}$ as the description of a predicate. In this way, we know the number of predicate descriptions that have short Hamming distances is bounded, in any Hamming ball.

Lastly, we give the definition of a Trevisan extractor.

Definition 3.2.7. *([42]) For $\mathscr{S} = (S_1, ..., S_{m'})$, $u \in \{0,1\}^n$ from a δ-source X, y uniformly random from $\{0,1\}^d$, and $\bar{u} = EC_{n,\sigma}(u)$ as the description of a predicate $\{0,1\}^l \to \{0,1\}$, we define a Trevisan extractor $\{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{m'}$ as $EXT_{\mathscr{S}}(u,y) = NW_{\mathscr{S},\bar{u}}(y) = \bar{u}(y|_{S_1})...\bar{u}(y|_{S_{m'}})$.*

Definition 3.2.8. *([42]) $EXT : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{m'}$ is a* strong (δ,ϵ)*-extractor if for every distribution X on $\{0,1\}^n$ of min-entropy δ, the induced distribution $(U_d, EXT(X,U_d))$ on $\{0,1\}^d \times \{0,1\}^{m'}$ has statistical distance at most ϵ from $U_d \times U_{m'}$.*

Remark 3.2.2. *A strong (δ,ϵ)-extractor is equivalent to an (n,m,d,δ,ϵ)-extractor, with $m = d + m'$.*

For the ease of understanding, we have the structure of the strong extractor and all of the relevant parameters plotted in Fig. 3.2. The strong extractor EXT takes input strings u from a δ-source and y as the seed, and outputs m' bits.

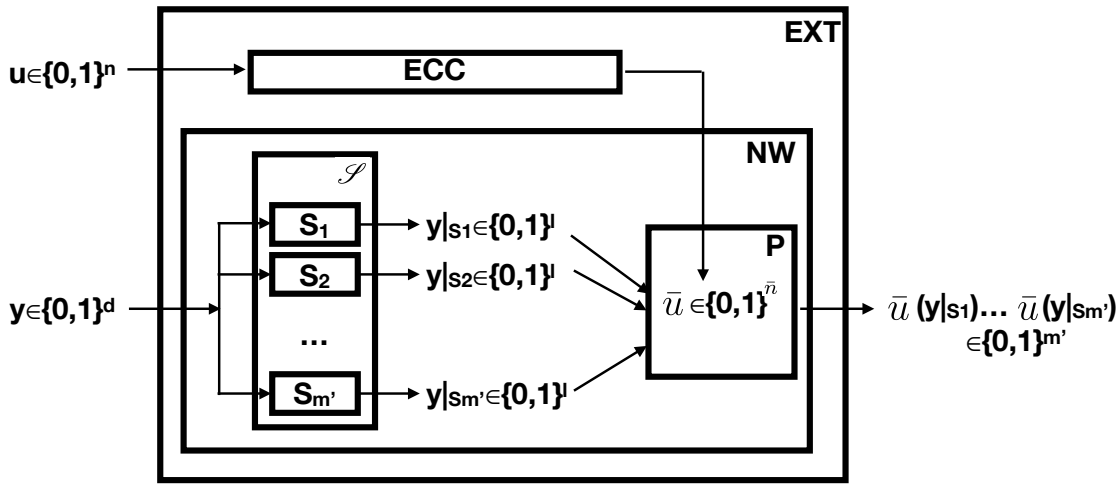

Figure 3.2: The structure of the strong randomness extractor EXT, which is composed of an error correcting code ECC and a Nisan-Wigderson (NW) generator. NW uses $\mathscr{S}$ as a weak design and the string encoded by ECC as a predicate P.

3.3 An NW Generator Is a Strong Extractor

In this section, we will prove the following theorem, which is improved from Proposition 10 in Raz et al. [42]. Specifically, the required parameter $\rho = \frac{\delta n - 3\log(m'/\epsilon) - d - 3}{m'}$

for extractors is improved to $\rho = \frac{\delta n - 3\log(m'/\epsilon) - (d-l) - 3 + \log 3}{m'-1}$. We can interpret the improvement as resulting in a slightly smaller δ value, which means the number of bad committees in the corresponding approximating disperser will be smaller, according to Proposition 2.3.1.

Theorem 3.3.1. *If $\mathscr{S} = (S_1, ..., S_{m'})$ (with $S_i \subset [d]$) is a weak (l, ρ)-design for $\rho = \frac{\delta n - 3\log(m'/\epsilon) - (d-l) - 3 + \log 3}{m'-1}$, then $EXT_{\mathscr{S}}$ defined in Definition 3.2.7 is a strong (δ, ϵ)-extractor, which is equivalent to an $(n, m' + d, d, \delta, \epsilon)$-extractor.*

Proof. The high level idea of the proof is that we first convert the problem of the extractor's output being far away from uniform to the problem that there exists a next-bit predictor that predicts the next bit with noticeable probability advantage. By utilizing the properties of error correcting codes, we are able to bound the probability that a next-bit predictor predicts the next bit. Therefore, if we can show that there is no predictor having noticeable probability advantage of guessing the next bit, then the output of the extractor is not far away from a uniform distribution.

From Yao [50], we know that if $\langle U_d, Z\rangle$ is a distribution on $\{0,1\}^d \times \{0,1\}^{m'}$ and the statistical difference of $\langle U_d, Z\rangle$ from $U_d \times U_{m'}$ is greater than ϵ, then there is an $i \in [m']$ and a function ("next-bit predictor") $A : \{0,1\}^d \times \{0,1\}^{i-1} \to \{0,1\}$ such that

$$\Pr_{\langle y,z\rangle \sim \langle U_d, Z\rangle}[A(y, z_1 z_2 ... z_{i-1}) = z_i] > 1/2 + \epsilon/m' \tag{3.1}$$

Therefore, to prove that the statistical distance of $\langle U_d, EXT_{\mathscr{S}}(X, U_d)\rangle$ from $U_d \times U_{m'}$ is at most ϵ, it is equivalent to prove that for every next-bit predictor,

$$\Pr_{\langle y,u\rangle \sim \langle U_d, X\rangle}[A(y, \bar{u}(y|_{S_1}) ... \bar{u}(y|_{S_{i-1}})) = \bar{u}(y|_{S_i})] \leq 1/2 + \epsilon/m', \tag{3.2}$$

where X is a distribution with min-entropy ratio δ and $\bar{u} = EC_{n,\sigma}(u)$.

If we fix a weak (l, ρ)-design $\mathscr{S} = (S_1, ..., S_{m'})$, all the bits of y outside S_i do not affect the prediction probability in Eq. (3.2). Thus, we first fix those $d-l$ bits outside S_i and analyze how many bits it takes to describe a predictor function A, and then we add $d - l$ bits on top of the analysis result.

We first analyze when $d - l$ bits of y outside S_i are fixed. Then $\bar{u}(y|_{S_1})$ only depends on $|S_1 \cap S_i|$ bits and can be described by a truth table with $2^{|S_1 \cap S_i|}$ bits. Similarly, for any $j < i$, $\bar{u}(y|_{S_j})$ can be described by a truth table with $2^{|S_j \cap S_i|}$ bits. Therefore, $\bar{u}(y|_{S_1}) ... \bar{u}(y|_{S_{i-1}})$ can be described by $\sum_{j<i} 2^{|S_i \cap S_j|}$ bits, which is no more than $\rho(m' - 1)$ bits, according to Definition 3.2.2. Furthermore, considering the $d - l$

bits we fixed for this analysis, a next-bit predictor function A can be described with no more than $d-l+\rho(m'-1)$. That means, there could be up to $2^{d-l+\rho(m'-1)}$ next-bit predictor functions. We denote the size of a family $\mathcal{A}$ of next-bit predictor functions as $|\mathcal{A}| = 2^{d-l+\rho(m'-1)}$.

We define $\mathcal{B}$ as the set of strings $u \in \{0,1\}^n$ for which there exists a next-bit predictor $A \in \mathcal{A}$ such that the relative Hamming distance of $\bar{u} = EC_{n,\sigma}(u)$ from A is within $1/2 - \epsilon/(2m')$, which is equivalent to A predicts the next bit with probability greater than $1/2 + \epsilon/(2m')$. By Definition 3.2.6 and Remark 3.2.1, we know that there would not be too many such bad u's. Specifically, let $\sigma = \epsilon/(2m')$, by the union bound, we have

$$|\mathcal{B}| \leq \frac{4m'^2}{3\epsilon^2}|\mathcal{A}| \leq \frac{4m'^2}{3\epsilon^2}2^{d-l+\rho(m'-1)} \tag{3.3}$$

Having $\sigma = \epsilon/(2m')$ means we will need an error correcting code $EC_{n,\sigma}(u)$ whose minimum relative Hamming distance is (at least) $1/2 - \epsilon^2/(8m'^2)$.

By Definition 3.2.7, u is selected from a δ-source and thus each u has probability at most $2^{-\delta n}$ of occurring, therefore, if we let $\rho = \frac{\delta n - 3\log(m'/\epsilon) - (d-l) - 3 + \log 3}{m'-1}$, then we have

$$\begin{aligned}
\Pr[u \in \mathcal{B}] &\leq \frac{4m'^2}{3\epsilon^2}2^{d-l+\rho(m'-1)} \times 2^{-\delta n} \\
&= \frac{4m'^2}{3\epsilon^2}2^{d-l+\delta n-3\log(m'/\epsilon)-(d-l)-3+\log 3} \times 2^{-\delta n} \\
&= \epsilon/(2m')
\end{aligned} \tag{3.4}$$

By the definition of $\mathcal{B}$, we know that when $u \notin \mathcal{B}$, $\Pr_{y\sim U_d}[A(y, \bar{u}(y|_{S_1})...\bar{u}(y|_{S_{i-1}})) = \bar{u}(y|_{S_i})] \leq 1/2 + \epsilon/(2m')$.

Eventually, we have

$$\begin{aligned}
\Pr_{\langle y,u\rangle\sim\langle U_d,X\rangle}[A(y, \bar{u}(y|_{S_1})...\bar{u}(y|_{S_{i-1}})) = \bar{u}(y|_{S_i})] &\leq \Pr_{u\sim X}[u \in \mathcal{B}] \cdot 1 + \Pr_{u\sim X}[u \notin \mathcal{B}] \cdot (\frac{1}{2} + \frac{\epsilon}{2m'}) \\
&\leq \frac{\epsilon}{2m'} + (\frac{1}{2} + \frac{\epsilon}{2m'}) \\
&= \frac{1}{2} + \frac{\epsilon}{m'}
\end{aligned} \tag{3.5}$$

As we have shown that for every next-bit predictor, the probability of predicting the next bit is not more than $1/2 + \epsilon/m'$. Thus, the statistical distance of

$\langle U_d, EXT_{\mathscr{S}}(X, U_d)\rangle$ from $U_d \times U_{m'}$ is at most ϵ. Therefore, $EXT_{\mathscr{S}}$ is a strong (δ, ϵ)-extractor. According to Definition 3.2.8, this is equivalent to an $(n, m' + d, d, \delta, \epsilon)$-extractor. □

3.4 Construction

In this section, we discuss the construction for the strong extractor $EXT_{\mathscr{S}}$, before which we construct a weak (l, ρ)-design and also an error correcting code $EC_{n,\sigma}(u)$ whose minimum relative Hamming distance is at least $1/2 - \epsilon^2/(8m'^2)$, as we mentioned in the proof in the previous section.

Construction of a Weak Design

It is shown in Raz et al. [42] that for every l, $m' \in \mathbb{N}$ and $\rho > 1$, there exists a weak (l, ρ)-design $S_1, ..., S_{m'} \subset [d]$ with $d = \lceil \frac{l}{\ln \rho} \rceil l$. Moreover, such a family can be found in time $poly(m', d)$.

The proof for this statement is to first show the existence of such a weak design via probabilistic analysis. Then authors in [42] derandomize the construction.

To be more specific, in order for showing the existence of a weak design, we separate $[d]$ into l blocks $B_1, ..., B_l$, each of size $\lceil l/\ln \rho \rceil$. Then a design $S_i = \{a_1, ..., a_l\}$ has $a_1, ..., a_l$ selected from $B_1, ..., B_l$ respectively, uniformly and independently. In this way, elements of S_i are independent. Further probabilistic analysis shows $\mathbb{E}[\sum_{j<i} 2^{|S_i \cap S_j|}] \leq \rho(i-1)$. Thus, with nonzero probability, there is a weak design such that $\sum_{j<i} 2^{|S_i \cap S_j|} \leq \rho(i-1)$.

To derandomize, authors in [42] show by averaging argument that there exists an element α_1 from the first block B_1 such that $\mathbb{E}[\sum_{j<i} 2^{|S_i \cap S_j|} | a_1 = \alpha_1] \leq \rho(i-1)$. Similarly for block 2 all the way up to block l, then we will have $\mathbb{E}[\sum_{j<i} 2^{|S_i \cap S_j|} | a_1 = \alpha_1, ..., a_l = \alpha_l] \leq \rho(i-1)$. By doing this, we have a set S_i for any $i \leq m'$. Then we will be able to construct a weak design.

Remark 16 in [42] states that it also works if S_i is chosen uniformly from all subsets of $[d]$ of size l, rather than dividing into blocks $B_1, ..., B_l$.

As the quality of the randomized construction for weak designs is *checkable* through the value of ρ. In practice, we fix m', d and l, and then randomly generate sets $S_1, ..., S_{m'} \subset [d]$ of size d. By doing so, we can calculate the value of ρ and we have a weak (l, ρ)-design. After repetitions, we pick the weak design that has the most

suitable ρ value.

Construction of an Error Correcting Code

For finding a desired error correcting code, we are specifically looking for binary codes. Non-binary codes like Reed-Solomon (RS) codes [43] may not be suitable for us, because Reed-Solomon codes operate on symbols from a finite field $\mathbb{F}_q$ rather than bits. The minimum relative Hamming distance we obtain from Reed-Solomon codes is different from the minimum relative Hamming distance we want for bit strings. Further, as Reed-Solomon codes usually have large alphabet sizes (i.e., $|\sum| = q$), it may perform poorly if we try to convert the minimum relative Hamming distance for Reed-Solomon codes into the minimum relative Hamming distance for bit strings, because two different symbols may only differ one single bit.

Binary Hadamard codes [34] become an option for us, because the codewords have minimum relative Hamming distance of $1/2$, which is greater than $1/2 - \epsilon^2/(8m'^2)$ as we needed. However, the price is that the length of codewords is as large as 2 to the power of the length of messages, i.e., $\bar{n} = 2^n$. Large $\bar{n}$ will result in large l, due to $l = \log \bar{n}$. Further, it results in a large value of d for the weak design construction. Recall that $D = 2^d$ is the size of a committee, which we would like to keep it from being too large.

The trade-off we can make here is that we sacrifice a little minimum relative Hamming distance from Hadamard codes in exchange for a much smaller length of codewords. Specifically, we break the input message into multi-bit symbols and encode each symbol via Hadamard codes. Reed-Solomon codes also naturally come into place as they can ensure two different input messages have a certain number of different symbols after encoding via Reed-Solomon codes. Thus, we use a concatenated error correcting code by concatenating a Reed-Solomon code with a Hadamard code. This kind of code is suggested by Trevisan [48] and Raz et al. [42]. In this way, the concatenated code can have minimum relative Hamming distance arbitrarily close to $1/2$, while still being a binary code as a whole and not having codeword length that is too large.

To describe this concatenated code in detail, we denote a Reed-Solomon code as having k input symbols, each of which has $\log q$ bits, and the number of output symbols is usually the same as the size of $\mathbb{F}_q$. Thus, a Reed-Solomon code can be denoted as $\{0,1\}^{k \log q} \rightarrow \{0,1\}^{q \log q}$, with minimum relative Hamming distance

$(q-k+1)/q$ [43].

As for Hadamard codes, the number of output bits is 2 to the power of the number of input bits. We denote a Hadamard code we will use as $\{0,1\}^{\log q} \to \{0,1\}^q$. That means each symbol the Reed-Solomon code outputs will be taken as an input for the Hadamard code. The minimum relative Hamming distance for the Hadamard code is 1/2.

Overall, we can denote the concatenated code as $\{0,1\}^{k \log q} \to \{0,1\}^{q \log q} \to \{0,1\}^{q^2}$, with minimum relative Hamming distance as $\frac{q-k+1}{2q}$. It is because any two different codewords in a Reed-Solomon code have at least $q-k+1$ difference symbols. After a Hadamard code, they will differ at least $(q-k+1)\frac{q}{2}$ bits. Lastly, we normalize the distance, resulting in the minimum relative Hamming distance of $\frac{(q-k+1)\frac{q}{2}}{q^2} = \frac{q-k+1}{2q} = \frac{1}{2} - \frac{k-1}{2q}$, which can be arbitrarily close to 1/2.

Construction of an Extractor

Now that we have a weak design and an error correcting code constructed, it suffices to construct our extractor as defined in Definition 3.2.7.

3.5 An Example

In this section, we will give an example of an application scenario to select committees of sizes 2^{55} from a network of size 2^{57}. The following procedure is very similar to Section 2.5.

We use a concatenated code of Reed-Solomon code and Hadamard code as our error correcting code, with $n = k \log q$ and $\bar{n} = q^2$. As for a weak (l, ρ)-design, where $l = \log \bar{n}$, we find that ρ has to be large to keep $d = \lceil \frac{l}{\ln \rho} \rceil l$ small, if we use the construction in Raz et al. [42]. However, large ρ will result in large δ, as we have $\rho = \frac{\delta n - 3\log(m'/\epsilon) - (d-l) - 3 + \log 3}{m'-1}$ according to Theorem 3.3.1. Further, we know we want to keep δ small, because in an approximating disperser, the number of bad committees is $K = 2^{\delta n+1}$, according to Proposition 2.3.1. Thus, we find it beneficial to randomly generate weak designs and apply the one with the smallest ρ value. As the weak design is constructed at the beginning of the system and fixed afterwards, we can construct it randomly at the beginning and distribute it to all nodes in the network as a part of the program.

We let $n = 2624$ as the number of input bits for the error correcting code, with

$k = 164$, $\log q = 16$ and $q = 2^{16}$. Thus, this error correcting code outputs $\bar{n} = q^2 = 2^{32}$ bits as codewords and has minimum relative Hamming distance as $\frac{1}{2} - \frac{k-1}{2q} = \frac{1}{2} - \frac{163}{2^{17}}$. Thus, $l = \log \bar{n} = 32$. We also let $d = 55$, $m' = 2$ and $\epsilon = 0.2$. We have the error correcting code we needed as we required the minimum relative Hamming distance to be at least $1/2 - \epsilon^2/(8m'^2) = 0.49875$. Instead of having $\rho \approx e^{l^2/d} = 121835437.376$ via the construction of the weak design in Raz et al. [42], we randomly generate weak designs for $1,000,000$ times[1] and pick the design with the smallest value of ρ, which is 1024. According to Theorem 3.3.1, when $\epsilon = 0.2$, we will need $\delta n \geq 1059$, the required number of truly random bits in an n-bit input string to the extractor. Thus, we have a $(2624, 57, 55, 0.40358, 0.2)$-extractor, which is equivalent to an $(N = 2^{2624}, M = 2^{57}, D = 2^{55}, K = 2^{1060}, \epsilon = 0.2)$-approximating disperser.

Now that we have an approximating disperser, we can proceed to the procedure of selecting the next committees. Assume that the number of Byzantine nodes in the network is $1/3 - \epsilon = 2/15$ so that committees has $2/3$ members as good nodes.

As our committee size is 2^{55}, while we only need 2624 bits as input for the approximating disperser, we again apply the Feige's algorithm to have each committee member contributes a random bit. We need $\lceil 2^{55}/2624 \rceil$ bins for the lightest bin to have at most 2624 bits. In expectation, each bin has 1749 good bits. By the Chernoff bound, we know the probability that a given bin has less than 1300 good bits is $2^{-91.4}$. After union bound, the probability that any of the bins has less than 1300 good bits is $2^{-47.8}$.

Similar to Eq. (2.8), under the condition that $\beta_n n = 2624 - 1300 = 1324$, the probability that we fail to select a good committee is $2^{1324}\frac{2^{1060}}{2^{2624}} = 2^{-240}$.

Overall, the failure probability is $1 - (1 - 2^{-47.8})(1 - 2^{-240}) \approx 2^{-47.79}$.

We again summarize what we have achieved in the above example, compared with what we aim for, shown in Table 3.2. We have BA_D in the table because we use it as a primitive for BA. Thus, the cost for time and messages can vary, depending on the implementation for BA_D. For more related works on time and message complexity on BA, please see Section 1.4. Number of rounds needed to select the next committee is at least $2 + BA_D$ because there will be one additional round for committee members to send out their choices of bit strings to other committee members and one final round for committee members to send out the consensus to non-committee nodes. The size of messages is approximately $D \cdot BA_D$ because we will do BA in parallel for each member from the committee of size D. Regarding the required storage space,

[1] This execution can be run on any typical computer and finished within one or two minutes.

each node in the network needs a program to do Reed-Solomon encoding, a generator matrix (16 by 2^{16}) for Hadamard encoding, and also a weak design (2 arrays, each of which has 32 integers). These sum up to around 0.3 MB storage.

Table 3.2: Parameters that we achieved for the PRG-based construction

Parameters	Target	Achieved
Network size (M)	$1,000,000$	2^{57}
Committee size (D)	$1,000$	2^{55}
Number of rounds each epoch	$10 \sim 20$	$2 + BA_D$
Size of messages received per committee node per epoch	2 GB	$D \cdot BA_D$
Required storage space per node	128 GB	0.3 MB
Byzantine ratio over the network (β)	1/3	2/15
Failure probability	2^{-40}	$2^{-47.49}$

From Table 3.2, we can see that the example we gave requires only a small amount of storage space and achieves the targeted failure probability and a reasonable Byzantine ratio. However, the required network size and committee size are rather large. The problem this may cause is that the time and the number of messages needed to select the next committee (or to reach BA) will also be large. Further, after the committee agrees on an input string to the disperser, it also takes a long time [2] for each node to compute the next committee, although this problem might be mitigated if the computation can be done in a distributed manner.

3.6 Numerical Results

To further illustrate how the performance of randomized weak designs is, compared with deterministic weak designs in Raz et al. [42], we conducted numerical experiments for showing what ρ values we can obtain from these two constructions. For the randomized construction, we randomly generate weak designs for $1,000,000$ times for each setting and pick the weak design with the smallest ρ value. For deterministic construction, we again set $\rho \approx e^{l^2/d}$ [42]. In numerical experiments, we set the value of m' to be small. For example, $m' = 2$ as in the previous section.

In the first experiment, we set $d = 55$. The value of l varies from 10 to 45. Since the value of ρ is too large, we set y-axis as $\log_2(\rho)$ instead of ρ, as shown in Fig. 3.3.

[2] Our current estimation is about 45 years to execute the computation on a normal laptop. In this case, local computation is not negligible.

We observe that random weak designs can obtain much smaller ρ value, compared with the deterministic ones.

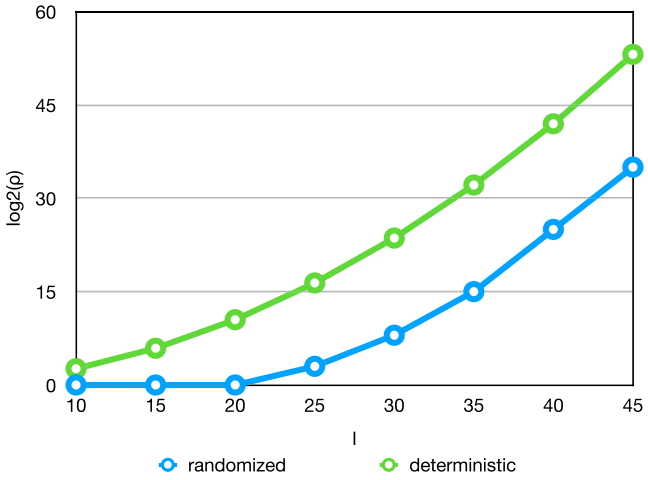

Figure 3.3: $d = 55$.

Similarly, we generate Fig. 3.4 by setting $l = 10$ and let d vary from 15 to 55. In this figure, the y-axis is ρ.

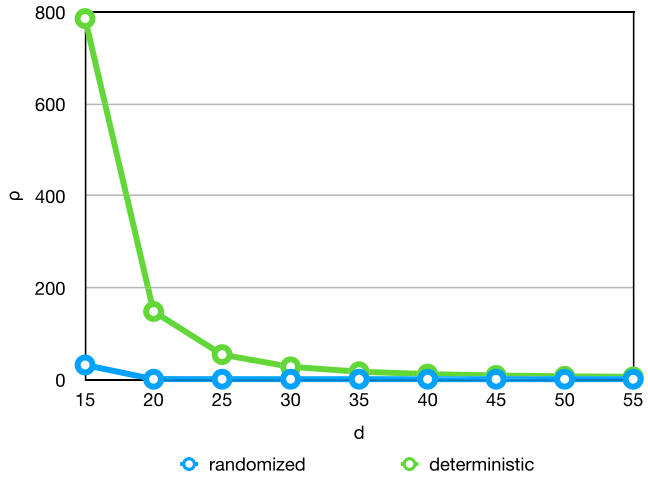

Figure 3.4: $l = 10$.

We therefore conclude randomized weak designs possess much better performance when compared with deterministic ones, especially when the value of m' is small.

Chapter 4

Random Walks on an Expander

In this chapter, we look into the construction of an extractor through random walks on expanders. However, we show how we cannot give a guarantee that the next committee will stay as good as the current committee, if the bits to describe the random walks come from the current committee. Nevertheless, it is still possible if those bits come from the whole network, which we will leave as a future work.

One key property for random walks on expanders is that those random walks resemble independent sampling, which is exactly what we needed. Because the naive solution to form a good committee is to randomly select nodes from the network. Unfortunately, we will show that this method does not work for our scenarios. Before that, we will first explain this method and show how specifically it is relevant to committee selection.

The reason why we are interested in random walks on expanders is that we can use the nodes on the random walks of length $D-1$ as the members in the committee, whose size is D if we include the initial node of the walks. It is one of the properties of random walks on expanders that those nodes on the path resemble independent sampling. We will analyze how close the resemblance is in the following. One more preliminary needed is that $n = m+(D-1)\log d_e$, where we use n as the number of bits input to the expander for the description of random walks, m as the number of bits needed to specify a node (i.e., the initial node of the walk), D as the size of committees, and d_e as the degree of the regular expander. The equation $n = m + (D-1)\log d_e$ can be interpreted as using m bits to specify the initial node and then take $D-1$ steps of walk, each of which needs $\log d_e$ to specify which neighbour is the next step.

Lemma 4.1. *Given bits describing the random walk of length $D-1$ on a d_e-regular*

expander from a $(1-\beta_n)$-source and β_c is the ratio of bad nodes in the committee, then the probability that the next committee formed by a random walk on the expander is bad is bounded by $2^{\beta_n\lceil m+(D-1)\log d_e\rceil+D}(\beta+\lambda_2)^{\lceil D\beta_c\rceil-1}$, where λ_2 is the second largest absolute eigenvalue of the transition matrix[1] of the expander.

Proof. Our proof for this lemma is divided into three steps. We first analyze random walks on an expander while assuming those walks are uniform random. In the last step (step 3), we will relax this constraint and use weak random source to provide bits describing the random walk.

1. Let I_{bad} be the set of indices of Byzantine nodes, on the path of the random walk. $I_{\text{bad}} \subset \{0,1,2,3,...,D-1\}$. For example, $I_{\text{bad}} = \{0,3,8,...\}$ means on the path of random walk on an expander, we encounter Byzantine nodes on step 0, step 3 and step 8, etc. According to Theorem 3.10 in Hoory et al. [19], for any fixed I_{bad}, we have

$$\Pr[\forall i \in I_{\text{bad}} : v_i \in B] \leq (\beta+\lambda_2)^{|I_{\text{bad}}|-1}, \tag{4.1}$$

   where B is the set of Byzantine nodes in the network. [2]

2. If an I_{bad} indicates a bad committee, namely $|I_{\text{bad}}| > D\beta_c$, we call it a bad I_{bad}. For each specified bad I_{bad}, the probability of that happens is always no more than $(\beta+\lambda_2)^{\lceil D\beta_c\rceil-1}$, regardless of $|I_{\text{bad}}|$, which ranges from $\lceil D\beta_c\rceil$ to D. Note that $\beta_c < 1/3$ is the ratio of bad nodes in the current committee and also the ratio that we target for the next committee to have. Plus, we know that there are no more than 2^D possible choices of I_{bad}, which is within 2-approximate [3] of the true number of bad I_{bad}'s, due to the symmetry of binomial coefficients. Given a perfect random string for a random walk, we define the event of selecting a bad committee as E_{badCom}. We have

$$\Pr[E_{\text{badCom}}] \leq 2^D(\beta+\lambda_2)^{\lceil D\beta_c\rceil-1} \tag{4.2}$$

[1] A transition matrix of a regular expander is obtained by dividing each entry of the adjacency matrix of the expander by the degree.

[2] As for why λ_2 plays an important role for random walks on expanders to converge to uniform samplings, audience may refer to Lemma 3.4 in [19] and its proof. The key to understand the proof might be to see the multiplication of a matrix and a vector as a linear transformation of the space, in the eigen-basis.

[3] Because $\beta_c < 1/3 < 1/2$ and the symmetry of binomial coefficients, $\binom{D}{D\beta_c} + \binom{D}{D\beta_c+1} + ... + \binom{D}{D-1} + \binom{D}{D} \geq 2^D/2$

3. Now let us relax the restriction that we have a perfect random source. In fact, our string is only from a weak random source that we defined as $(1-\beta_n)$-source. We claim that the probability of electing a bad committee will grow by at most $2^{n\beta_n}$, similar to the analysis for Corollary 2.2.1. Define E_{fail} as the event that we select a bad committee using $(1-\beta_n)$-source. By union bound, we have

$$\begin{aligned}\Pr[E_{\text{fail}}] &\leq 2^{n\beta_n}\Pr[E_{\text{badCom}}] \\ &\leq 2^{\beta_n(m+(D-1)\log d_e)+D}(\beta+\lambda_2)^{\lceil D\beta_c\rceil-1}\end{aligned} \tag{4.3}$$

□

In Lemma 4.1, we give a bound on the probability that a random walk on an expander outputs a bad committee. However, we will soon realize if the n-bit description of the random walk comes from the current committee, then we will not be able to give any guarantee that the next committee can be as good as the current one, regardless of what the value of β_c is.

One assumption we need to reach this negative result is that the tightest bound we can give for the second largest absolute eigenvalue of the d_e-regular expander is the Ramanujan bound. Specifically, that means the maximum of the absolute values of all the non-trivial eigenvalues is $2\sqrt{d_e-1}$ (Lubotzky et al. [32] and Murty [36]), for the adjacency matrix of the expander. Moreover, if we divide every entry in the adjacency matrix by degree d_e, we obtain a transition matrix and $\lambda_2 \leq \frac{2\sqrt{d_e-1}}{d_e}$.

For the construction of a d_e-regular expander, one simple randomized construction is to have $d_e/2$ uniform random permutations on the set of nodes (M). It is conjectured by Alon [2] and proved by Friedman [13] that a random d_e-regular graph is "almost Ramanujan", in the sense that for every constant $c > 0$, $\lambda_2 < \frac{2\sqrt{d_e-1}+c}{d_e}$, asymptotically almost surely. In 2015, Puder [41] provided a new and substantially simpler proof that a random d_e-regular graph satisfies $\lambda_2 < \frac{2\sqrt{d_e-1}+1}{d_e}$, asymptotically almost surely.

Assuming Ramanujan bound ($\lambda_2 \leq \frac{2\sqrt{d_e-1}}{d_e}$), we are now ready to show the negative result:

Lemma 4.2. *Regardless of β_c, the bound in Lemma 4.1 cannot give any guarantee that the next committee is as good as the current one.*

Proof. In our application scenario, the bits describing the random walk come from the current committee. Specifically, each good current member contributes random $\log d_e$

bits, concatenates each member's bits together and adds $n - D\log d_e = m + (D-1)\log d_e - D\log d_e = m - \log d_e$ 0's as padding. Thus, the ratio of non-random bits in the description is no less than the ratio of bad nodes in the current committee, namely $\beta_n \geq \beta_c$. Also, assume $\log d_e \leq m$. Then we have $2^{\beta_n \lceil m+(D-1)\log d_e \rceil} \geq 2^{\beta_c D \log d_e}$. Then the bound in Lemma 4.1 is:

$$\begin{aligned} 2^{\beta_n \lceil m+(D-1)\log d_e \rceil + D}(\beta+\lambda_2)^{\lceil D\beta_c \rceil - 1} &\geq 2^{\beta_c D \log d_e + D}\left(\beta + \frac{2\sqrt{d_e-1}}{d_e}\right)^{\lceil D\beta_c \rceil - 1} \\ &> 2^{\beta_c D \log d_e + D}\left(\frac{2\sqrt{d_e-1}}{d_e}\right)^{D\beta_c} \\ &> (d_e)^{\beta_c D}\, 2^D \left(\frac{1}{d_e}\right)^{D\beta_c} \\ &= 2^D \\ &> 1 \end{aligned} \tag{4.4}$$

□

Audience may start wondering whether the bound given in Eq. (4.1) is too loose to give any guarantee. However, we again reach a negative result when using a Chernoff bound (Theorem 1.1 in Healy [18]):

Let G be a d_e-regular expander on W and fix a sequence of functions $f_i : W \to [0,1]$ each with mean $\mu_i = \mathbb{E}_w[f_i(w)]$. If we consider a random walk w_1, ..., w_D on G, then for all $c > 0$,

$$\Pr[|\sum_{i=1}^{D} f_i(w_i) - \sum_{i=1}^{D} \mu_i| \geq cD] \leq 2e^{-\frac{c^2(1-\lambda_2)D}{4}} \tag{4.5}$$

Similar with the analysis in Section 2.2, we define a function f that outputs 1 if the input is an ID corresponding to a Byzantine node. $\mu = \beta$ is the expectation probability of a node being Byzantine. We then proceed to prove the following lemma:

Lemma 4.3. *Regardless of β_c, the bound in Eq. (4.5) cannot give any guarantee that the next committee is as good as the current one.*

Proof. This proof is similar as the proof for Lemma 4.1. We first assume random walks are uniform random in order to apply Eq. (4.5), and then we relax this constraint to allow weak random source.

We define variable X as the number of Byzantine nodes on a random walk of length $D-1$, and we fail if $X > \beta_c D$. The probability that we fail, using Eq. (4.5), is bounded:

$$\begin{aligned}
\Pr[X > \beta_c D] &\leq \Pr[X \geq \beta_c D] \\
&= \Pr[X - \beta D \geq (\beta_c - \beta) D] \\
&\leq \Pr[|X - \beta D| \geq (\beta_c - \beta) D] \\
&\leq 2e^{-\frac{(\beta_c - \beta)^2 (1-\lambda_2) D}{4}}
\end{aligned} \tag{4.6}$$

If n bits describing the random walk are from the current committee, we again name this source as $(1-\beta_n)$-source, where $\beta_n \geq \beta_c$ and $n = m + (D-1)\log d_e$. Thus, the failure probability will grow by at most $2^{\beta_n n}$. Define the failure event as E_{fail}, then $\Pr[E_{\text{fail}}] \leq 2^{\beta_n n} \Pr[X > \beta_c D]$.

$$\begin{aligned}
2^{\beta_n n} \Pr[X > \beta_c D] &= 2^{\beta_n (m + (D-1)\log d_e)} 2e^{-\frac{(\beta_c - \beta)^2 (1-\lambda_2) D}{4}} \\
&\geq 2^{\beta_c D \log d_e} 2e^{-\frac{(\beta_c - \beta)^2 (1-\lambda_2) D}{4}} \\
&\geq 2^{\beta_c D \log d_e} e^{-\frac{\beta_c^2 (1-\lambda_2) D}{4}} \\
&\geq 2^{\beta_c D \log d_e - 0.36 \beta_c^2 (1-\lambda_2) D} \\
&\geq 2^{\beta_c D (\log d_e - 0.36 \beta_c (1-\lambda_2))} \\
&> 1
\end{aligned} \tag{4.7}$$

The last step in Eq. (4.7) is due to $d_e > 2$, so that $\log d_e > 1$ while $0.36\beta_c(1-\lambda_2) < 1$. Therefore, we are not able to give a guarantee for random walks on an expander to output a committee as good as the current one.

□

Note that in this proof, we make no assumption on the value of λ_2, other than being in the range of 0 and 1.

Chapter 5

Conclusions and Future Works

Our goal for this thesis was to look into the practical aspects of committee selection algorithms, under the full information model, while making no cryptographic assumptions except that a message receiver knows who the message sender is. The solution can be used as a sub-routine to achieve BA against a semi-adaptive adversary with sub-quadratic message complexity, and can also be used in scalable blockchain protocols. Our key to the solution is to utilize approximating dispersers. With an approximating disperser and given a good current committee, we are able to select the next good committee with high probability, even when the input source for the approximating disperser is not perfectly random.

Theoretically, there exist numerous methods for constructing approximating dispersers (or equivalently randomness extractors). We first showed what would be the best we can do through a fully randomized construction for approximating dispersers and developed our Byzantine Agreement algorithm accordingly. At first glance, the randomized construction seemed to be promising for solving our problem. Such an approximating disperser could be constructed at the beginning of the system and embedded in the program distributed to the nodes in the network; however, the description of the randomized approximating disperser is too large to be stored and distributed onto today's commercial devices.

We then discussed the PRG-based construction, improved some constants, and also made a small part of the construction (the weak designs) randomized. This results in a provably good construction with a small description.

Random walks on an expander is one of the construction methods. However, we proved that if the descriptions for random walks come from the current committee, then there is no guarantee that the next committee will stay as good as the current

one.

The following problems are left as future works:

- This thesis work can likely be extended to selecting M good committees, which can be used in designing a both scalable and load-balanced BA protocol, without assuming a trusted committee at the beginning of the system, see King et al. [29, 25];
- To address the storage problem raised by the fully randomized construction for approximating dispersers, we wonder whether it is possible to use distributed storage among M good committees?
- Again on the storage issue, can we prove that commonly used pseudo-random number generators could be helpful to construct fully randomized approximating dispersers with high probability?
- If the descriptions for random walks on an expander comes from the entire network, can we guarantee the next committee stays as good as the current one?
- One issue omitted in this thesis is that we did not consider the case where the number of active nodes is less than M. For example, nodes may actively join and leave the network, so that some IDs may not correspond to active nodes anymore. However, this problem could be addressed in a load balanced manner by referring to King et al. [25, 3].

Bibliography

[1] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous Byzantine agreement with expected $O(1)$ rounds, expected $O(n^2)$ communication, and optimal resilience. In *International Conference on Financial Cryptography and Data Security*, pages 320–334. Springer, 2019.

[2] Noga Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

[3] John Augustine, Valerie King, Anisur R Molla, Gopal Pandurangan, and Jared Saia. Scalable and secure computation among strangers: Resource-competitive Byzantine protocols. *arXiv preprint arXiv:1907.10308*, 2019.

[4] Boaz Barak, Yevgeniy Dodis, Hugo Krawczyk, Olivier Pereira, Krzysztof Pietrzak, François-Xavier Standaert, and Yu Yu. Leftover hash lemma, revisited. In *Annual Cryptology Conference*, pages 1–20. Springer, 2011.

[5] Mihir Bellare, Oded Goldreich, and Madhu Sudan. Free bits, PCPs, and non-approximability—towards tight results. *SIAM Journal on Computing*, 27(3):804–915, 1998.

[6] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[7] Nicolas Braud-Santoni, Rachid Guerraoui, and Florian Huc. Fast Byzantine agreement. In *Proceedings of the 2013 ACM symposium on Principles of distributed computing*, pages 57–64, 2013.

[8] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[9] Ran Canetti, Oded Goldreich, and Shai Halevi. The Random oracle methodology, revisited. *Journal of the ACM (JACM)*, 51(4):557–594, 2004.

[10] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for Byzantine agreement. *Journal of the ACM (JACM)*, 32(1):191–204, 1985.

[11] Uriel Feige. Noncryptographic selection protocols. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 142–152. IEEE, 1999.

[12] Michael J Fischer and Nancy A Lynch. A lower bound for the time to assure interactive consistency. Technical report, GEORGIA INST OF TECH ATLANTA SCHOOL OF INFORMATION AND COMPUTER SCIENCE, 1981.

[13] Joel Friedman. A proof of Alon's second eigenvalue conjecture and related problems. *arXiv preprint cs/0405020*, 2004.

[14] Oded Goldreich. A note on computational indistinguishability. *Information Processing Letters*, 34(6):277–281, 1990.

[15] Oded Goldreich and Hugo Krawczyk. Sparse pseudorandom distributions. In *Conference on the Theory and Application of Cryptology*, pages 113–127. Springer, 1989.

[16] Venkatesan Guruswami, Christopher Umans, and Salil Vadhan. Unbalanced expanders and randomness extractors from Parvaresh–Vardy codes. *Journal of the ACM (JACM)*, 56(4):1–34, 2009.

[17] Johan Håstad, Russell Impagliazzo, Leonid A Levin, and Michael Luby. Construction of a pseudo-random generator from any one-way function. In *SIAM Journal on Computing*. Citeseer, 1993.

[18] Alexander D Healy. Randomness-efficient sampling within NC. *Computational Complexity*, 17(1):3–37, 2008.

[19] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006.

[20] Russell Impagliazzo, Leonid A Levin, and Michael Luby. Pseudo-random generation from one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 12–24. ACM, 1989.

[21] Yaoqi Jia, Tarik Moataz, Shruti Tople, and Prateek Saxena. Oblivp2p: An oblivious peer-to-peer content sharing system. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 945–962, 2016.

[22] Jesse Kamp and David Zuckerman. Deterministic extractors for bit-fixing sources and exposure-resilient cryptography. *SIAM Journal on Computing*, 36(5):1231–1247, 2006.

[23] Bruce M Kapron, David Kempe, Valerie King, Jared Saia, and Vishal Sanwalani. Fast asynchronous Byzantine agreement and leader election with full information. *ACM Transactions on Algorithms (TALG)*, 6(4):68, 2010.

[24] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for Byzantine agreement. In *Annual International Cryptology Conference*, pages 445–462. Springer, 2006.

[25] Valerie King, Steven Lonargan, Jared Saia, and Amitabh Trehan. Load balanced scalable Byzantine agreement through quorum building, with full information. In *International Conference on Distributed Computing and Networking*, pages 203–214. Springer, 2011.

[26] Valerie King and Jared Saia. From almost everywhere to everywhere: Byzantine agreement with $\widetilde{O}(n^{3/2})$ bits. In *International Symposium on Distributed Computing*, pages 464–478. Springer, 2009.

[27] Valerie King and Jared Saia. Scalable Byzantine computation. *ACM SIGACT News*, 41(3):89–104, 2010.

[28] Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: Scalable Byzantine agreement with an adaptive adversary. *Journal of the ACM (JACM)*, 58(4):18, 2011.

[29] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 990–999. Society for Industrial and Applied Mathematics, 2006.

[30] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Towards secure and scalable computation in peer-to-peer networks. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 87–98. IEEE, 2006.

[31] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[32] Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[33] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30. ACM, 2016.

[34] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*, volume 16. Elsevier, 1977.

[35] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge University Press, 2017.

[36] M Ram Murty. Ramanujan graphs. *Journal-Ramanujan Mathematical Society*, 18(1):33–52, 2003.

[37] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, 2008.

[38] Noam Nisan and Amnon Ta-Shma. Extracting randomness: A survey and new constructions. *Journal of Computer and System Sciences*, 58(1):148–173, 1999.

[39] Noam Nisan and David Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996.

[40] David Peleg. Distributed Computing. *SIAM Monographs on discrete mathematics and applications*, 5:1–1, 2000.

[41] Doron Puder. Expansion of random graphs: New proofs, new results. *Inventiones mathematicae*, 201(3):845–908, 2015.

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

[43] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[44] Peter Robinson, Christian Scheideler, and Alexander Setzer. Breaking the $\widetilde{\Omega}(\sqrt{n})$ barrier: Fast consensus under a late adversary. *arXiv preprint arXiv:1805.00774*, 2018.

[45] Rüdiger Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Proceedings First International Conference on Peer-to-Peer Computing*, pages 101–102. IEEE, 2001.

[46] Ronen Shaltiel. Recent developments in explicit constructions of extractors. In *Current Trends in Theoretical Computer Science: The Challenge of the New Century Vol 1: Algorithms and Complexity Vol 2: Formal Models and Semantics*, pages 189–228. World Scientific, 2004.

[47] Aravind Srinivasan and David Zuckerman. Computing with very weak random sources. *SIAM Journal on Computing*, 28(4):1433–1459, 1999.

[48] Luca Trevisan. Extractors and pseudorandom generators. *Journal of the ACM*, 48(4):860–879, 2001.

[49] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151(2014):1–32, 2014.

[50] Andrew C Yao. Theory and application of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 80–91. IEEE, 1982.

[51] David Zuckerman. Randomness-optimal oblivious sampling. *Random Structures & Algorithms*, 11(4):345–367, 1997.

[52] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 681–690. ACM, 2006.