Network Extractor Protocols

Yael Tauman Kalai
Microsoft Research
yael@microsoft.com

Xin Li [*]
University of Texas at Austin
lixints@cs.utexas.edu

Anup Rao [†]
Institute for Advanced Study
arao@ias.edu

David Zuckerman [‡]
University of Texas at Austin
diz@cs.utexas.edu

Abstract

We design efficient protocols for processors to extract private randomness over a network with Byzantine faults, when each processor has access to an independent weakly-random n-bit source of sufficient min-entropy. We give several such network extractor proto-cols *in both the information theoretic and computational settings.*

For a computationally unbounded adversary, we construct protocols in both the synchronous and asynchronous settings. These network extractors imply efficient protocols for leader election (synchronous setting only) and Byzantine agreement which tolerate a linear fraction of faults, even when the min-entropy is only $2^{(\log n)^{\Omega(1)}}$. For larger min-entropy, in the synchronous setting the fraction of tolerable faults approaches the bounds in the perfect-randomness case.

Our network extractors for a computationally bounded adversary work in the synchronous setting even when 99% of the parties are faulty, assuming trapdoor permutations exist. Further, assuming a strong variant of the Decisional Diffie-Hellman Assumption, we construct a network extractor in which all parties receive private randomness. This yields an efficient protocol for secure multi-party computation with imperfect randomness, when the number of parties is at least $\mathrm{polylog}(n)$ and where the parties only have access to an independent source with min-entropy $n^{\Omega(1)}$.

[*]Supported in part by NSF Grant CCF-0634811 and THECB ARP Grant 003658-0113-2007.

[†]Supported in part by NSF Grants CCF-0634811 and CCR-0324906. Part of this work was done while the author was visiting Microsoft Research at Redmond.

[‡]Supported in part by NSF Grant CCF-0634811.

1 Introduction

Randomization appears extremely useful in designing algorithms for a variety of problems. However, many researchers believe $\mathrm{BPP} = \mathrm{P}$, in which case randomness is not needed for efficient algorithms. In contrast, randomness is provably necessary to solve many problems in distributed computing and cryptography. For example, randomness is necessary for Byzantine agreement in the asynchronous setting, and necessary to do efficiently in the synchronous setting.

These applications of randomness typically require long strings of uniformly random bits, yet it is unclear how to obtain such random strings. Instead, we may only have access to a weak source: an unknown, arbitrary distribution with some entropy. We use a standard measure of entropy called *min-entropy*: a distribution has min-entropy k if all strings have probability at most 2^{-k}. When such a distribution is over n-bit strings, we refer to it as an (n, k)-source.

Much work has been devoted to simulating randomized algorithms using weak sources with small min-entropy [26, 8, 27, 25, 2], culminating with Andreev et al.'s simulation of any BPP algorithm with an $(n, n^{\Omega(1)})$-source.

Goldwasser, Sudan, and Vaikuntanathan [16] addressed the natural question of whether weak sources can be used in distributed computing. They showed how to solve Byzantine agreement even if each processor has a weak source, and these weak sources are independent. However, they considered fairly restricted weak sources, and asked whether similar results hold for general (n, k)-sources. In this paper, we answer this question positively, showing strong results for general sources. Our methods also give strong results for leader election and other distributed computing problems.

Several researchers considered the analogous question in the cryptographic context [19, 18, 12, 10, 7]. A few positive results emerged; however, arguably the most impressive-sounding results are the negative ones of Dodis et al. [11]. They showed that almost all of the classic cryptographic tasks, including encryption, bit commitment, secret sharing, and secure two-party computation (for nontrivial functions), are impossible even with a single $(n, .99n)$-source.

Despite this, under a strong variant of the Decisional Diffie-Hellman (DDH) Assumption, we give a protocol for secure multi-party computation when the number of parties is at least $\mathrm{polylog}(n)$, even if each processor only has access to an independent $(n, n^{\Omega(1)})$-source. A simpler version of our protocol works in the honest-but-curious setting for a constant number of parties.

1.1 Network Extractors

The standard method for handling weak sources is to use a randomness extractor: a function which converts a weak source into a nearly-random string. While this is possible for certain restricted weak sources, it is not for general (n, k)-sources. However, it is possible with the addition of a few truly random bits, called a seed [20]. Even in a world with no truly random bits, such seeded extractors are sometimes useful; for example, in algorithms we can cycle over all possibilities for the seed and take a majority vote. However, in distributed computing and cryptography, this idea appears less useful.

Instead, the proper tool for distributed computing is a *network extractor protocol*, which was introduced (but left unnamed) by Dodis and Oliveira [10] in the cryptographic context and by Goldwasser et al. [16] in the context of Byzantine agreement. The idea is that each processor starts with a weak source, and these weak sources are independent. The processors then interact, and at the end of the protocol most, if not all, non-faulty processors obtain private, nearly-random strings. These strings may then be used in any traditional protocol requiring perfect randomness.

Before elaborating, we discuss the model that we consider. We assume that p processors communicate with each other via point-to-point channels in order to perform a task. However, an unknown t of the processors are *faulty*. We allow Byzantine faults: faulty processors may behave arbitrarily and even collude maliciously. We call the set of faulty processors the *adversary*, and we only consider a non-adaptive adversary – the set of faulty processors is fixed in advance and does not change. We assume that the communication channels are not private, so the adversary can see all communication. This is called the *full information model.* We note that we could obtain stronger results in a network with private channels, however in the interest of space we focus on the full information model.

Most of our results are for *synchronous* networks: communication between processors takes place in rounds and every message transmitted at the beginning of a round is guaranteed to reach its destination at the end of the round. In this case we allow rushing: the faulty processors may wait for all good processors to transmit their messages for a particular round, before transmitting their own messages. We also have results for *asynchronous* networks– here the only guarantee is that every message will eventually be received.

We assume each processor has access to an unknown, arbitrary (n, k)-source of randomness, and that these sources are independent. This independence assumption seems justifiable if we view the processors as being physically far away from each other. Such sources may also arise if the adversary manages to acquire (say via a virus) a small amount of information about each of the honest processors' (truly random) sources. In this case, conditioning on the adversary's information leaves each of the processors with independent weak sources. Indeed, the model we consider seems to be the most natural generalization of the model where each processor has access to truly random bits.

In order to define network extractor, we need some notation. Processor i begins with a sample from a weak source $x_i \in \{0,1\}^n$ and ends in possession of a hopefully uniform sample $z_i \in \{0,1\}^m$. Let b be the concatenation of all the messages that were sent during the protocol. Capital letters such as Z_i and B denote these strings viewed as random variables.

We define network extractors both in the information-theoretic setting (where the adversary may be computationally unbounded) and in the computational settings (where the adversary is assumed to be computationally bounded), starting with the former.

Definition 1.1 (Network Extractor)**.** A protocol for p processors is a (t, g, ϵ) *network extractor* for min-entropy k if for any min-entropy k independent sources $X_1, \ldots, X_p$ over $\{0,1\}^n$ and any choice of t faulty processors, after running the protocol, the number of processors i for which $|(B, Z_i) - (B, U_m)| < \epsilon$ is at least g. Here U_m is the uniform distribution on m bits, independent of B, and the absolute value of the difference refers to variation distance. We say that a protocol is a *synchronous extractor* if it is a network extractor that operates over a synchronous network. We say that it is an *asynchronous extractor* if it is a network extractor that operates over an asynchronous network.

Let $\mathcal{G} = \{i_1, \ldots, i_g\}$ denote the set of processors with private, random outputs: $|(B, Z_i) - (B, U_m)| < \epsilon$. Because each Z_i depends only on X_i and B, the above condition implies that

$$|(B, (X_i)_{i \notin \mathcal{G}}, (Z_i)_{i \in \mathcal{G}}) - (B, (X_i)_{i \notin \mathcal{G}}, U_{gm})| < g\epsilon.$$

In other words, after running the network extractor protocol, the joint distribution of the outputs of all the processors in $\mathcal{G}$ is indistinguishable from independent uniform strings, even after seeing all communication and all the sources of the rest of the processors.

We next define a *computational* network extractors. Here we assume that all the processors involved (honest and faulty) are computationally bounded (i.e., run in time $\mathrm{poly}(n)$), and the outputs need only be computationally (rather than statistically) indistinguishable from uniform. We restrict our attention to the synchronous setting.

Definition 1.2 (Computational Network Extractor)**.** A protocol for p processors is a (t, g) *computational network extractor* for min-entropy k if for any min-entropy k independent sources $X_1, \ldots, X_p$ over $\{0,1\}^n$ and any choice of t faulty processors, with probability $1 - \mathrm{negl}(n)$, after running the protocol there are g honest processors $\mathcal{G} = \{i_1, \ldots, i_g\}$ such that

$$\{(B, (X_i)_{i \notin \mathcal{G}}, (Z_i)_{i \in \mathcal{G}}\}_{n \in \mathbb{N}} \approx \{B, (X_i)_{i \notin \mathcal{G}}, U_{gm}\}_{n \in \mathbb{N}}$$

where U_{gm} is the uniform distribution on gm bits, independent of B and $(X_i)_{i \notin \mathcal{G}}$, and where $\approx$ denotes computational indistinguishability.

A priori, it is not clear that network extractors, information-theoretic or computational, even exist.

2 Our Results

We give several constructions of network extractors, both in the information-theoretic setting and in the computational setting.

2.1 Network Extractors in the Information-Theoretic Setting

All of our network extractors in the information theoretical setting have the additional property that the g processors with private, random outputs can be named in advance, in the sense that there are $g + t$ processors known before the start of the protocol, such that any non-faulty processor among them will obtain a private, random output.

As long as the min-entropy rate of the sources is greater than $1/2$, we give nearly optimal network extractors. In particular, as long as the fraction of faulty processors t is bounded by a constant less than 1, we show how to build a one round synchronous network extractor which leaves almost every non-faulty processor with private randomness.

Theorem 2.1 (High Entropy Synchronous Extractor)**.** *For all $p, t, n, \alpha, \beta > 0$, there exists a constant $c = c(\alpha)$ and a two-round $(t, p - (1+\alpha)t - c, 2^{-k^{\Omega(1)}})$ synchronous extractor for min-entropy $k \geq (\frac{1}{2} + \beta)n$ in the full-information model. The protocol is one round for $t = \Omega(p)$.*

If the min-entropy of the general sources is much smaller, we can still design a good network extractor, though fewer processors end up with private random bits. The new protocol ensures roughly $p - (2 + \frac{\log\log n}{\log\log k})t$ honest processors end up with private randomness, thus tolerating a linear fraction of faulty processors. This protocol runs in a constant number of rounds even with min-entropy $k = 2^{(\log n)^{\Omega(1)}}$:

Theorem 2.2 (Low Entropy Synchronous Extractor)**.** *For all $p, t, \beta > 0$, $k > \log p$, and $n \leq 2^{O(t)}$, there exists a constant $c = c(\beta)$ and a $(1/\beta + 1)$ round $(t, p - (1.1 + 1/\beta)t - c, 2^{-k^{\Omega(1)}})$ synchronous extractor for min-entropy $k \geq 2^{\log^\beta n}$ in the full-information model.*

In the asynchronous setting, we get slightly weaker results:

Theorem 2.3 (High Entropy Asynchronous Extractor)**.** *For all $p, t, n, \beta > 0$, there exists a one-round $(t, p - 3t - 1, 2^{-k^{\Omega(1)}})$ asynchronous extractor for min-entropy $k \geq (\frac{1}{2} + \beta)n$ in the full-information model.*

Theorem 2.4 (Low Entropy Asynchronous Extractor)**.** *There exist constants $c_1, c_2 > 0$ such that for all $p, t, \beta > 0$, $k > \log p$, and $\mathrm{poly}(t) \leq n \leq 2^{O(t)}$, there exists a $(1/\beta + 1)$ round $(t, p - c_1 t/\beta - c_2, 2^{-k^{\Omega(1)}})$ asynchronous extractor for min-entropy $k \geq 2^{\log^\beta n}$ in the full-information model.*

2.2 Byzantine Agreement and Leader Election with Weak Random Sources

After running a network extractor, we can run any of the traditional distributed computing protocols designed for perfect, private randomness. We now describe applications of our results to two basic distributed computing problems: Byzantine agreement and leader election/collective coin-flipping.

Byzantine Agreement. The goal of a protocol for *Byzantine agreement* is for the processors to agree on the result of some computation, even if some t of them are faulty. Byzantine agreement is fundamental because it can be used to simulate broadcast and maintain consistency of data.

For protocols using weak sources, the only results are due to Goldwasser et al. [16]. They require all weak sources to have min-entropy rate at least $1/2$. In the full information model, they only obtain results for weak sources that are more restricted than block sources[1]. Under the assumption that the processors have access to general (n, k)-sources, they give results only for the case of private channels. They posed the open question of whether protocols can be designed in the full information model assuming only that each processor has access to general (n, k)-sources. We answer this question positively by first running our network extractor, and then running a protocol for Byzantive Agreement with perfect randomness [15, 17].

In the synchronous setting, we essentially match the perfect-randomness case [15] when the min-entropy rate is greater than $1/2$, and we can tolerate a linear fraction of faults even with min-entropy $2^{(\log n)^{\Omega(1)}}$.

Theorem 2.5 (Synchronous Byzantine Agreement)**.** *Let $\alpha, \beta > 0$ be any constants. For p large enough, assuming each processor has access to an independent (n, k)-source, there exists synchronous $O(\log p)$ expected round protocols for Byzantine Agreement in the full information model with the following properties.*

1. *The protocol for $k \geq (1/2 + \beta)n$ tolerates $(1/3 - \alpha)p$ faulty processors.*
2. *The protocol for $k \geq n^\beta$ tolerates $(1/4 - \alpha)p$ faulty processors.*
3. *The protocol for $k \geq 2^{\log^\beta n}$ tolerates $p/(3.1 + 1/\beta)$ faulty processors.*

In the asynchronous case, we can tolerate a linear fraction of faults in only a polylogarithmic number of rounds, as is the case with perfect randomness [17].

Theorem 2.6 (Asynchronous Byzantine Agreement)**.** *Let $\alpha, \beta > 0$ be any constants. For p large enough, assuming each processor has access to an independent (n, k)-source, there exists a constant $0 < \gamma < 1$ and asynchronous $\mathrm{polylog}(p)$ expected round protocols for Byzantine Agreement in the full information model with the following properties.*

1. *The protocol for $k \geq (1/2 + \beta)n$ tolerates $(1/8 - \alpha)p$ faulty processors.*
2. *The protocol for $k \geq 2^{\log^\beta n}$ tolerates $\beta\gamma p$ faulty processors.*

Leader Election and Collective Coin Flipping. The goal of a protocol for *leader election* is to select a uniformly random leader from a distributed network of n processors. In the presence of faulty processors, we would like to bound the probability that one of the faulty processors gets selected as the leader.

Ben-Or and Linial [4] were the first to study collective coin-flipping under what we call the BL model: the full information model with reliable broadcast in a synchronous network. A long sequence of work [24, 1, 5, 9, 21, 28, 23, 13] has resulted in a protocol which tolerates $(1/2 - \alpha)p$ faulty processors and requires only $\log^*(p) + O(1)$ rounds to elect a leader (and hence perform a collective coin flip) in the BL model [23, 13]. We obtain essentially the same results if the processors' min-entropy rate is above 1/2, and we can tolerate a linear fraction of faults with min-entropy $2^{(\log n)^{\Omega(1)}}$.

Theorem 2.7 (Leader Election)**.** *Let $\alpha, \beta > 0$ be any constants. For p large enough, assuming each processor has access to an independent (n, k)-source, there exists $\log^* p + O(1)$ round protocols for leader election in the BL model with the following properties.*

1. *The protocol for $k \geq (1/2 + \beta)n$ tolerates $(1/2 - \alpha)p$ faulty processors.*
2. *The protocol for $k \geq n^\beta$ tolerates $(1/3 - \alpha)p$ faulty processors.*
3. *The protocol for $k \geq 2^{\log^\beta n}$ tolerates $(1/(2 + 1/\beta) - \alpha)p$ faulty processors.*

[1]They refer to these sources as block sources, though they are not as general as block sources are defined in this paper and the extractor literature.

2.3 Network Extractors in the Computational Setting

In the computational setting, under cryptographic assumptions, we are able to get network extractors for low min entropy rates where most, or even *all*, honest players get private randomness. However, this requires the number of players to be a growing parameter in terms of n. We focus on the synchronous full information model, where each player has an $(n, n^{\Omega(1)})$-source.

Our first result shows that if trapdoor permutations exist, then there exists a computational network extractor for sources with min-entropy $k = n^{\Omega(1)}$, in which almost every non-faulty processor ends up with a (computationally) private random string.

Theorem 2.8. *Assume that trapdoor permutations exist. Then for every $\alpha, \beta, \gamma > 0$ there exists a constant $0 < c < 1$ (that depends only on β) such that for every $\log^7 n \le p \le k^c$ there exists a $(t = \gamma p, p - (1 + \alpha)t)$ computational network extractor for min-entropy $k \ge n^\beta$ in the full information model.*

This matches the information theoretic guarantees with min-entropy $k > n/2$. Namely, using cryptographic assumptions we reduced our min-entropy requirement from $k > n/2$ to $k = n^{\Omega(1)}$ (though we added the requirement that the number of players is large enough in terms of n).

Our second result shows that under a (seemingly) stronger and non-standard cryptographic assumption, there exists a computational network extractor in which *every* non-faulty processor ends up with a (computationally) private random string. The assumption we rely on for this result is a strong variant of the DDH Assumption. We note that several strong variants of the DDH Assumption have been proposed earlier (eg., [6]). Our assumption, stated below, is weaker than some and is incomparable to others.

Strong DDH Assumption. For every security parameter n there exists an n-bit safe prime $p = 2q + 1$, and a generator $g \in Q_p$, such that $\{g^a, g^b, g^{ab}\}_{n \in \mathbb{N}} \approx \{g^a, g^b, g^c\}_{n \in \mathbb{N}}$, where $b, c \in_R \mathbb{Z}_q^*$, $a \in \mathbb{Z}_q^*$ is a random variable with min-entropy $n^{\Omega(1)}$, and all the operations are modulo p.

Theorem 2.9. *Assume the strong DDH Assumption holds. Then for every β, γ there exists a constant $0 < c < 1$ (that depends only on β) such that for every $\log^7 n \le p \le k^c$, there exists a $(t = \gamma p, p - t)$ computational network extractor for min-entropy $k \ge n^\beta$ in the full information model.*

Theorem 2.9 gives a protocol where each processor takes as input a weak source and *all* the (honest) processors end up with a (computationally) private random string. It is well known [14], that once each (honest) processor has a (computationally) private random string, they can securely compute any functionality (assuming the existence of enhanced trapdoor permutations, or alternatively, assuming the DDH Assumption). Thus, we get the following corollary.

Corollary 2.10. *Assume that the strong DDH Assumption holds. Then for every $\beta, \gamma > 0$ there exists a constant $0 < c < 1$ (that depends only on β) such that for every $\log^7 n \le p \le k^c$, the following holds: Every functionality (involving p processors) can be computed securely in the full information model with broadcast channels*[2]*, assuming that there are at most γp faulty processors, and assuming that each processor has access to an independent (n, n^β)-source.*

We note that all the results above (in the computational setting) assumed that the number of players p is a growing parameter. However, in many applications, for example that of secure multiparty computation with imperfect randomness, we would like to handle a constant number of players. We show that if the min entropy rate is greater than $1/2$, or if there is an honest majority, then under the strong DDH Assumption there are computational network extractors for *constant* number of players, where *all* honest players end up with private randomness.

Theorem 2.11. *Assume the strong DDH Assumption holds. For all $\alpha, \beta > 0$ there exists a constant $c = c(\alpha, \beta)$ such that for every $t \in \mathbb{N}$ and every integer $p \ge (1 + \alpha)t + c$, there exists a $(t, p - t)$ computational network extractor for min-entropy $k \ge (\frac{1}{2} + \beta)n$.*

Theorem 2.12. *Assume the strong DDH Assumption holds. For all $\alpha, \beta > 0$ there exists a constant $c = c(\alpha, \beta)$ such that for every $t \in \mathbb{N}$ and every integer $p \ge 2t + c$, there exists a $(t, p - t)$ computational network extractor for min-entropy $k \ge n^\beta$.*

We note that in the honest-but-curious model the problem becomes much easier. As Dodis et al. noticed [11], assuming the existence of enhanced trapdoor permutations, there is a simple protocol for secure two-party computation in the honest-but-curious model, assuming each party has access to an independent source

[2]Alternatively, instead of assuming broadcast channels, we can assume a trusted pre-processing phase for setting up a public-key infrastructure.

of min-entropy $k = (\frac{1}{2}+\beta)n$. In general, we show that if there is a strong C-source extractor with negligible error for (n, k)-sources, then under the Strong DDH Assumption, one can do secure multiparty computation with $C + 1$ or more players in the honest-but-curious model, where the players have independent (n, k)-sources.

3 Techniques

A useful tool here is a (strong) C-source extractor: an algorithm to extract randomness from C independent sources. Such an extractor immediately yields some kind of network extractor. We could designate p/C of the processors as receivers, and each receives the weak random strings from $\mathsf{C} - 1$ other processors. The processor can then apply the extractor to these $\mathsf{C} - 1$ strings plus its own. If an honest processor receives strings from only honest processors, then the output will be close to uniform.

We can do much better than this. The idea is to allow a processor to receive several C-tuples. We then apply a C-source extractor to each C-tuple (for now the received processor doesn't use its own string). If at least one of these C-tuples contain strings only from honest processors, then the output Y will be what's called a somewhere-random source. We think of Y as a matrix where each row corresponds to the output of the extractor on one C-tuple. Informally, a distribution on matrices is somewhere random if one of the rows in the matrix is distributed uniformly.

Barak, Rao, Shaltiel and Wigderson [3] gave a construction of a (strong) extractor for a somewhere random source with r rows and another $O(\frac{\log r}{\log k})$ independent (n, k) sources. As a special case, if the number of rows in the somewhere random source is small, then we need only one more independent source. By applying this BRSW extractor to the above matrix Y plus the processor's own string, the output will be close to uniform.

These are the underlying tools in all our constructions, both in the information theoretic setting and in the computational setting.

3.1 The Information Theoretic Setting

In the information-theoretic setting, we get a simple network extractor using the above ideas, assuming the min entropy k is a large enough polynomial in the number of parties p. In particular, first consider the case where $k = n^{\beta}$. The extractor for such sources [22, 3] requires only $\mathsf{C} = O(1/\beta)$ sources, which is a constant. If $t + \mathsf{C}$ players broadcast their strings, at least C of these players are honest. We can therefore get a somewhere-random source by applying our C-source extractor to each C-tuple. The BRSW extractor will work if the min-entropy k significantly exceeds the number of rows $\binom{t+\mathsf{C}}{\mathsf{C}}$. We use variants of expanders and dispersers to reduce this min-entropy requirement.

Our first ingredient is what we call an AND-disperser. This is a bipartite graph where for any set S of constant density on the right, there are a constant fraction of vertices on the left whose neighbors all lie in S.

We use such a graph with left degree C as follows. Each right vertex will represent a processor who announces its source (and hence won't end up with private randomness). There will be $(1 + \alpha)t$ right vertices, so at least αt of these processors must be honest. Each left vertex represents a string computed by applying a C-source extractor to the sources of its neighbors. We say that a left vertex is *good* if all its neighbors are honest, and thus its corresponding string is almost uniform. By the properties of the AND-disperser, at least a constant fraction of left vertices are good.

If we associated left vertices with processors, we would end up with only a constant fraction of such processors getting good randomness. Instead, we compose the AND-disperser with a bipartite expander graph. That is, we take an expander graph whose right vertices represent the strings obtained via the AND-disperser (so the right vertex set of the expander has the same size as the left vertex set of the AND-disperser), and whose left vertices represent all processors who didn't reveal their sources earlier. Each such processor is then associated with the strings corresponding to its neighbors. If at least one such neighbor is good, then these strings correspond to a somewhere-random source (with some small error). By the expansion property, many processors will have such a somewhere-random source.

Note that the number of rows of the somewhere random source is exactly the degree of the expander. Thus as long as this degree is small, the processor can use the BRSW extractor to extract private random bits from its own source using the somewhere random source. In the case where $\mathsf{C} = O(1/\beta)$ is a constant, there is a constant fraction of *good* left vertices of the AND-disperser and we can use a constant-degree bipartite expander. Therefore we obtain somewhere random sources with a constant number of rows, which is good enough for a processor to extract private random bits using its own source and the somewhere random source. This is our one-round protocol, which works for min-entropy $k = n^{\beta}$.

For smaller min-entropy, the fraction of *good* left vertices of the AND-disperser will be sub-constant, and the expanders won't have constant degree. As a result the BRSW extractor no longer works with just one additional independent source. Therefore, we will need multiple rounds to extract randomness. We do this by noting that the BRSW extractor will work with $\mathsf{C}' > 1$ additional independent sources. This is the same situation as in the beginning, except now we need $\mathsf{C}' < \mathsf{C}$ independent sources to extract random bits. Thus we repeat the above one-round protocol, but in place of the independent source extractor we use the BRSW extractor.

The key observation is that in each round the number of rows of the somewhere random source will decrease, and hence so will the number of required independent sources. It turns out that this number decreases quickly (by a factor of $\Omega(\log k)$ in each round), so we only need to repeat the one-round protocol for a small number (roughly $\frac{\log \log n}{\log \log k}$) of rounds before an honest processor can use only its own source and the somewhere random source to extract private random bits. In each round $(1 + \alpha)t$ new processors send out their strings. This gives us the general synchronous extractor.

The asynchronous extractor uses the same ideas discussed above, plus additional ideas to deal with the asynchrony. First we need more processors to send out their strings. A simple example is that in the synchronous case if a processor receives strings from $t+1$ processors then at least one string is from an honest processor. In the asynchronous case we must let the processor receive strings from $2t + 1$ processors, so that it will eventually receive $t + 1$ strings and at least one string is from an honest processor. Similar issues apply to obtaining at least one C-tuple which consists of all honest processors. We give a new AND-disperser to deal with this, which guarantees that the fraction of "good" tuples is bigger than that of "bad" tuples.

Next, instead of using an expander graph, we use an extractor graph, which guarantees the right fraction of good and bad tuples in the neighbors for most processors. This ensures that the argument discussed above works for most processors.

However, in the asynchronous setting we have two more problems. First, even with the AND-disperser and the extractor graph, there will still be a small fraction of unlucky honest processors left who could wait forever in the protocol. Second, without synchronization, the faulty processors can wait to see the honest processors' strings and then make the somewhere random source dependent on the honest processors' sources. Before describing our solution, we note the structure of our r-round protocol. Define a *round-i processor* as one which announces its source in the ith round, and a round-$r+1$ processor never announces its source. All messages in the synchronous protocol are passed from round i processors to round $i + 1$ processors.

We use a simple message transmitting mechanism to deal with both problems above. Specifically, whenever a round-i processor receives enough strings from round-$(i - 1)$ processors to form a new somewhere random source, it sends out "Complete i" to all other round-i processors. A processor aborts if it receives many "Complete i" messages before it receives enough strings from round-$(i - 1)$ processors, in which case it also broadcasts "Complete i". We show that eventually every processor in round i will receive many "Complete i" messages; thus no processor will wait forever, and the fraction of honest processors who don't get a somewhere random source only increases slightly. Moreover, each round-i processor only announces its source after it has received many "Complete i" messages. Thus before even a single honest round-i processor announces its source, a large fraction of round-i processors have already finished computing the new somewhere random sources. For these processors, the somewhere random sources must be independent of the processors' own sources (since the processors have not announced their sources yet).

3.2 The Computational Setting

We now focus on the case where each player has an independent (n, k) source, for $k = n^{\beta}$. The information theoretic protocols for this setting of parameters start with a large set of players (more than t players) revealing their sources. This immediately results in a significant loss in the number of players that end up with private randomness. Namely, the guarantee is that only $p - 2t$ players end up with private randomness. Thus, this network extractor is meaningful only in the case of honest majority.

In the computational setting, we construct a protocol in which only a *small* set of honest players (much smaller than t) reveal their sources. At first this seems to be useless, since it may be the case that all the players who reveal their sources are malicious. However, we note that in this case, the fraction of honest players has increased among the set of players who haven't yet revealed their sources. We use this fact to our advantage. More specifically, the players reveal their sources in some pre-specified order, guaranteeing that in some round, several honest players must reveal their sources simultaneously. At first it seems hard to take advantage of this situation, since we do not know who the honest players are and cannot identify a round in which honest players revealed their sources. Nonetheless, we manage to make progress under computational assumptions.

We construct a protocol that proceeds in d rounds, where in the j'th round all the players in the j'th set (which is of size significantly smaller than t) announce their sources to all the players. The intuition is that if, in some round j, a sufficient number of honest players announce their sources then we are in good shape, since the rest of the players can use the announced strings to generate a somewhere random matrix: each row in the matrix corresponds to the output of a C-source extractor applied to C of the announced strings. On the other hand, if almost all of the players in the set are dishonest, we didn't lose much by having them announce their private sources, and it seems like we made some kind of progress since the fraction of honest players among the remaining players has gone up.

However, instructing all the players to announce their sources in the appropriate round is obviously not a good idea, since then by the end of the protocol, all of the honest players will have completely revealed their sources to the adversary. Yet, consider the *first* round j in which a significant number of honest players announce their sources. At the end of this round, only a small set of honest players have announced their sources, and every honest player who hasn't announced her source can obtain a private random string! We use this fact to our advantage.

We use computational assumptions to change the protocol in order to ensure that after this "good" round the honest players do not reveal any additional information about their source (to a computationally bounded adversary). Instead of instructing each player to announce her source in the appropriate round, we instruct each player to announce a *function* f of her source. On the one hand, this function f should maintain the entropy of the source (i.e., should be injective). On the other hand, f should hide all information about the source. These two requirements, of being injective and hiding, can be achieved simultaneously only in the computational setting, under cryptographic assumptions. Moreover, in order to hide all information about the source, the function f needs to be randomized.

Note that at the end of the "good" round, many of the players have a random string. So, one could try using part of this (supposedly) random string as randomness for the function. Unfortunately, this will not work since each of the random strings possessed by the players depends on their private sources, and for the function f to be hiding the random string should be independent of the sources. We need one more idea to overcome this obstacle.

Recall that in every round (in particular the "good" round), a set of strings are announced, and each player uses the announced strings to try to extract a random (uniformly distributed) string from her private source. Each player first saves a chunk of her (supposedly) random string as private randomness, where at the end of the protocol all these d chunks (one chunk per each round) will be xored and will consist the player's output. Then, all the players use a small fresh chunk of these (supposedly) random strings to generate for each player P_i a private random string *independent* of the source x_i. This is done as follows: First, each player stretches her small chunk of randomness using a pseudo-random generator (which is known to exist assuming the existence of one-way functions). Then, for each player P_i, all players use a portion of the chunk to run a coin flipping protocol, in which only player P_i receives a random string r_i. Now each player P_i has a (supposedly) private random string r_i, which is independent of her source x_i, and will continue to the next round of the protocol, while using $f(x_i, r_i)$ as her private source.

However, this is still not good enough, since for the proof to work we need to ensure that each random string r_i is independent of *all* the sources simultaneously. To this end, we instruct all the players to use a small fresh chunk of their random strings to elect a small set of leaders. Each of these leaders will no longer use their private sources in subsequent rounds, and will use the all-zero string instead. Now, if there is at least one honest leader, then each r_i is independent of *all* the remaining sources simultaneously. Finally, in order to ensure that with high probability, at least one of the leaders is indeed honest, we need to assume that the number of players is large enough (this is where we use the assumption that the number of players is growing in terms of n).

This is the high-level idea of the proof of Theorem 2.8. Note that in the above protocol (which we will refer to as the initial protocol) several players don't end up with private randomness. In particular, the players that announced their source before the "good" round may not get private randomness since, at the point of announcement, the (randomized) function of their source may actually reveal significant information about their source. This is the case since the string used as randomness in the function may actually not be random at all (as randomness is not available yet). Similarly, the leaders who were elected before the "good" round may not get private randomness. However, the number of rounds d is small (in particular, is $o(p)$), the number of players who reveal their source in each round is small (in particular, is $o(p)$), and the number of leaders elected in each round is very small. Thus, we can ensure that the number of honest player that *do not* end up with private randomness is small, and is $o(p)$ according to our setting of parameters (which is significantly smaller than the number of dishonest players t, which is assumed to be a constant fraction of the total number of players).

There is another technicality that we overlooked. The initial protocol as described above, needs a broadcast channel (or alternatively, a public-key infrastructure) to execute the coin-flipping and leader election protocols. Intuitively, a broadcast channel is needed to agree on the output.

Looking closely at the protocol, we notice that we do not actually need a broadcast channel for our coin flipping protocols, since in each of these protocols only a *single* player receives an output. So, to run these coin-flipping protocols all we need is to instruct all the players to send this player a non-malleable commitment to a random string, and then reveal the random string. This player will then use the xor all these random strings as her private output.

On the other hand, the leader election protocol seems to really need a broadcast channel. To eliminate this need, we change the protocol yet again. Instead of running a single leader election protocol per round, in which all players need to agree on who the leaders are, we run p protocols in each round (one protocol per player), where in the i'th protocol only player P_i receives an output. If the output is 1 then player P_i thinks of herself as elected as leader, and if the output is 0 then she thinks of herself as a non-leader. Each of these protocols will output 1 only with small probability; loosely speaking if we want to elect ℓ leaders per round, then each of these protocols will output 1 with probability ℓ/p. The players will use a fresh chunk of their (supposedly) random string for each of these protocols.

For Theorem 2.9, the high-level idea is to start by running the initial protocol, described above, in order to generate (computationally) private random strings to *some* of the players. The players will then use these strings to generate (computationally) private random strings for *all* the players. When doing this, we distinguish between players that were elected as leaders, and players that were not elected as leaders.

A player that was elected as leader never announces her source (or a function of her source) in the initial protocol. Moreover, we set the parameters so that at the end of the initial protocol, her source still has high min-entropy conditioned on all the messages that were sent during the protocol. The idea is for the leaders to use a seeded extractor to extract randomness from their sources. To obtain a random-looking seed R, after running the initial protocol, all the players will run a secure coin-flipping protocol. Then each leader will apply a (strong) seeded extractor to her source, while using R as the seed.

For non-leader players, their sources may not have any min-entropy after running the initial protocol above. So, instead, we would like the leaders (who now possess private random-looking strings) to give them a fresh chuck of their random-looking string. However, to gain privacy this needs to be done over a secure channel. So, we would like to "generate" secure channels. For this we use the strong DDH Assumption.

Each player, rather than announcing a (randomized) function of her source X_i in the initial protocol, will announce a function of $(g^{X_i} \mod p)$, where p is some fixed (safe) prime and g is some fixed element in $\mathbb{Z}_p^*$. We think of $(g^{X_i} \bmod p)$ as the player's public-key and of X_i as her secret key. After the leaders extract private randomness, each leader will partition her random string to sufficiently many parts,[3] and will send each non-leader an encrypted fresh chuck of her private randomness, using El-Gamal encryption. This scheme is known to be semantically secure under the DDH Assumption. However, since in our case X_i is not random, but only has high min-entropy (and since p and g are a priori fixed rather than randomly chosen), we need to rely on a stronger variant of the DDH Assumption.

Finally, the players will also add a zero-knowledge proof-of-knowledge of their plaintext to ensure non-malleability. This, yet again, introduces technical problems that we need to deal with: Zero-knowledge proofs require both the prover and the verifier to be randomized, and at this point the verifier may not have a random string. We note that if the verifier does not possess private randomness then soundness is no longer guaranteed

We overcome this obstacle as follows: The zero-knowledge proofs we use have perfect completeness (i.e., an honest prover convinces an honest verifier of a true statement with probability 1), and the additional property that all the messages sent by the verifier are random, and consist of the verifier's random coin tosses (so they can be verified by anyone who sees the transcript). Since our verifiers may not have private randomness, they will ask the other players to generate the random messages on their behalf. However, since we do not

[3]As previously remarked, in the computational setting we can always expand the number of computationally random bits by using a pseudo-random generator.

know who are the honest players that have private randomness, each verifier will ask *all* players to verify each zero-knowledge proof. Namely, each zero-knowledge proof is now repeated p times, where each proof a different player sends the verifier's messages. The verifier accepts if *all* the transcripts are accepting ones.

We end by noting that the protocol described above can be seen in greater generality. Using the strong DDH Assumption, we can convert any network extractor (computational or information theoretical) into a computational network extractor in which *all* players end up with private randomness. However, we need the initial network extractor to have the property that after running it, there is a known subset of players S, which includes at least two honest players, such that all the honest players in S end up with private randomness.

The idea of the transformation is to first run the initial protocol using the sources $(g^{X_i} \bmod p)$, rather than X_i. Next, the players in S possessing private random strings give part of their random string to players outside of S. This is done by using the strong DDH assumption to "generate" secure channels, as above.

References

[1] N. Alon and M. Naor. Coin-flipping games immune against linear-sized coalitions. *SIAM Journal on Computing*, 22(2):403–417, Apr. 1993.

[2] A. E. Andreev, A. E. F. Clementi, J. D. P. Rolim, and L. Trevisan. Weak random sources, hitting sets, and BPP simulations. *SIAM Journal on Computing*, 28:2103–2116, 1999.

[3] B. Barak, A. Rao, R. Shaltiel, and A. Wigderson. 2 source dispersers for $n^{o(1)}$ entropy and Ramsey graphs beating the Frankl-Wilson construction. In *STOC'06*, 2006.

[4] M. Ben-Or and N. Linial. Collective coin flipping. *Randomness and Computation*, 1978.

[5] R. B. Boppana and B. O. Narayanan. Perfect-information leader election with optimal resilience. *SIAM J. Comput*, 29(4):1304–1320, 2000.

[6] R. Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *Advances in Cryptology — CRYPTO '97, 17th Annual International Cryptology Conference, Proceedings*, pages 455–469, 1997.

[7] R. Canetti, R. Pass, and A. Shelat. Cryptography from sunspots: How to use an imperfect reference string. In *FOCS'07*, pages 249–259, 2007.

[8] B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on Computing*, 17(2):230–261, 1988.

[9] J. Cooper and N. Linial. Fast perfect-information leader-election protocols with linear immunity. *Combinatorica*, 15(3):319–332, 1995.

[10] Y. Dodis and R. Oliveira. On extracting private randomness over a public channel. In *RANDOM 2003*, pages 252–263, 2003.

[11] Y. Dodis, S. J. Ong, M. Prabhakaran, and A. Sahai. On the (im)possibility of cryptography with imperfect randomness. In *FOCS'04*, pages 196–205, 2004.

[12] Y. Dodis and J. Spencer. On the (non)universality of the one-time pad. In *FOCS'02*, page 376. IEEE Computer Society, 2002.

[13] U. Feige. Noncryptographic selection protocols. In IEEE, editor, *FOCS'99*, pages 142–152, pub-IEEE:adr, 1999. IEEE Computer Society Press.

[14] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 218–229. ACM, 1987.

[15] S. Goldwasser, E. Pavlov, and V. Vaikuntanathan. Fault-tolerant distributed computing in full-information networks. In *FOCS*, pages 15–26. IEEE Computer Society, 2006.

[16] S. Goldwasser, M. Sudan, and V. Vaikuntanathan. Distributed computing with imperfect randomness. In P. Fraigniaud, editor, *19th International Symposium on Distributed Computing DISC 2005, Proceedings*, volume 3724 of *Lecture Notes in Computer Science*, pages 288–302. Springer, 2005.

[17] B. Kapron, D. Kempe, V. King, J. Saia, and V. Sanwalani. Fast asynchronous Byzantine agreement and leader election with full information. In *SODA'08*, pages 1038–1047, 2008.

[18] U. M. Maurer and S. Wolf. Privacy amplification secure against active adversaries. In *Advances in Cryptology — CRYPTO '97, 17th Annual International Cryptology Conference, Proceedings*, pages 307–321, 1997.

[19] J. L. McInnes and B. Pinkas. On the impossibility of private key cryptography with weakly random keys. In *Advances in Cryptology — CRYPTO '90, 10th Annual International Cryptology Conference, Proceedings*, pages 421–435, 1990.

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

[21] R. Ostrovsky, S. Rajagopalan, and U. Vazirani. Simple and efficient leader election in the full information model. In *STOC'94*, pages 234–242, 1994.

[22] A. Rao. Extractors for a constant number of polynomially small min-entropy independent sources. In *STOC'06*, 2006.

[23] A. Russell and D. Zuckerman. Perfect information leader election in log* n+O (1) rounds. *Journal of Computer and System Sciences*, 63(4):612–626, 2001.

[24] M. Saks. A robust noncryptographic protocol for collective coin flipping. *SIAM Journal on Discrete Mathematics*, 2(2):240–244, May 1989.

[25] M. Saks, A. Srinivasan, and S. Zhou. Explicit OR-dispersers with polylog degree. *Journal of the ACM*, 45:123–154, 1998.

[26] U. V. Vazirani and V. V. Vazirani. Random polynomial time is equal to slightly-random polynomial time. In *FOCS'85*, pages 417–428, 1985.

[27] D. Zuckerman. Simulating BPP using a general weak random source. *Algorithmica*, 16:367–391, 1996.

[28] D. Zuckerman. Randomness-optimal oblivious sampling. *Random Structures and Algorithms*, 11:345–367, 1997.