Pseudorandom Constructions: Computing in Parallel and Applications to Edit Distance Codes

by

Kuan Cheng

A dissertation submitted to The Johns Hopkins University

in conformity with the requirements for the degree of

Doctor of Philosophy

Baltimore, Maryland

May, 2019

Abstract

The thesis focuses on two problems about pseudorandom constructions.

The first problem is how to compute pseudorandom constructions by constant depth circuits. Pseudorandom constructions are deterministic functions which are used to substitute random constructions in various computational tasks. Constant depth circuits here refer to the computation model which can compute functions using circuits of AND, OR and negation gates, with constant depth, unbounded fan-in, taking function inputs by input wires and giving function outputs by output wires. They can be simulated by fast parallel algorithms. We study such constructions mainly for randomness extractors, secret sharing schemes and their applications. Randomness extractors are functions which transform biased random bits to uniform ones. They can be used to recycle random bits in computations if there are some entropies remaining. Secret sharing schemes efficiently share secrets among multi-parties s.t. the collusion of a bounded number of parties cannot recover any information of the secret while a certain larger number of parties can recover the secret. Our work constructs these objects with near optimal parameters and explores their applications.

The second problem is about applying pseudorandom constructions to build error correcting codes (ECCs) for edit distance. ECCs project messages to codewords in a metric space s.t. one can recover the codewords even if there are bounded number of

errors which can drive the codeword away by some bounded distance. They are widely used in both the theoretical and practical part of computer science. Classic errors are hamming errors which are substitutions and erasures of symbols. They are well studied by numerous literatures before. We consider one kind of more general errors i.e. edit errors, consists of insertions and deletions that may change the positions of symbols. Our work give explicit constructions of binary ECCs for edit errors with redundancy length near optimal. The constructions utilize document exchange protocols which can let two party synchronize their strings with bounded edit distance, by letting one party send a short sketch of its string to the other. We apply various pseudorandom constructions to get deterministic document exchange protocols from randomized ones. Then we construct ECCs using them. We also extend these constructions to handle block insertions/deletions and transpositions. All these constructions have near optimal parameters.

Thesis Committee

Primary Readers

Xin Li (Primary Advisor)

Assistant Professor

Department of Computer Science

Johns Hopkins University Whiting School of Engineering

Amitabh Basu

Associate Professor

Department of Applied Mathematics and Statistics

Johns Hopkins University Whiting School of Engineering

Michael Dinitz

Assistant Professor

Department of Computer Science

Johns Hopkins University Whiting School of Engineering

Abhishek Jain

Assistant Professor

Department of Computer Science

Johns Hopkins University Whiting School of Engineering

Acknowledgments

I am very grateful to my advisor Xin Li. Having his guidance is a great fortune. He introduced to me the wonderful research field centred around pseudorandomness and coding, helping me establishing the most important basics. During the proceedings of my research, I benefited a lot from his remarkable ideas and suggested reading materials. My first project started from his suggestion of reading locally computable pseudorandom generators. There were so many times that he provided amazing methods to help me stepping forward. He also supported me on participating various academic activities, from which I got chances to communicate and collaborate with famous researchers. In addition to academic, I also received precious recommendations and opportunities from him for my future career.

Besides my advisor, I am also grateful to other professors who taught, discussed with and advised me. Abhishek Jain taught me modern cryptography and advised me on my second project. Michael Dinitz taught me approximation algorithm and game theory s.t. I get the chance to use approximation techniques in one of my papers later. Vova Braverman taught me randomized algorithms which helped me to master the basic methods of randomness. Amitabh Basu gave several lectures and seminar talks in optimization and machine learning which trigger my interests in these related fields. Yuval Ishai initiated the discussion about secret sharing in $\mathcal{AC}^0$ which later became

my second paper. He also gave very important comments on my presentation in the conference. Also thank Bernhard Haeupler for joint work with me on edit distance topics.

Graduate students at Johns Hopkins University and other institutes also impact me a lot. I would like to thank my co-authors Ke Wu, Amirbehshad Shahrasbi and Zhengzhong Jin for joint work with me on topics of synchronization strings, document exchange and edit error ECCs. Also thank Yu Zheng for collaboration with me on edit distance related topics. Thank Zeyu Zhang for numerous discussions on various topics about academic, career, puzzle solving, etc., and for collaboration on several course objects. Also thank Andong Zhan and Qian Ke for discussions and collaborations on the machine learning course.

I also would like to thank Zaoxing Liu, Lin Yang, Tuo Zhao, Nikita Ivkin, Yasamin Nazari, Arka Rai Choudhury, Fabian Prada Nino, Sing Chun Lee, Venkata Gandikota, Xuan Wu, Chang Lou, Yigong Hu, Hang Zhu, Zhihao Bai, Cong Gao, Zhuolong Yu, Shiwei Weng, Hongyuan Mei, Weichao Qiu for discussions about work/life topics and having various entertaining activities together with me.

At last I want to thank my family for their all-the-time support. Also thank all my friends from other departments and associations for making Johns Hopkins a happy place to study and live.

My work is supported in part by NSF award CCF-1617713.

Table of Contents

Chapter 1

Randomness Exraction in $\mathcal{AC}^0$

1.1 Introduction

Randomness extractors are functions that transform biased random sources into almost uniform random bits. Throughout this chapter, we model biased random sources by the standard model of general weak random sources, which are probability distributions over n-bit strings with a certain amount of min-entropy k.[1] Such sources are referred to as (n,k)-sources. In this case, it is well known that no deterministic extractors can exist for one single weak random source even if $k = n-1$; therefore seeded randomness extractors were introduced in [NZ96], which allow the extractors to have a short uniform random seed (say length $O(\log n)$). In typical situations, we require the extractor to be *strong* in the sense that the output is close to uniform even given the seed. Formally, we have the following definition.

Definition 1.1.1 ([NZ96])**.** *A function* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *is a seeded*

[1] A probability distribution is said to have min-entropy k if the probability of getting any element in the support is at most 2^{-k}.

(k, ϵ) *extractor if for any* (n, k) *source* X*, we have*

$$|\mathsf{Ext}(X, U_d) - U_m| \leq \epsilon.$$

Ext *is strong if in addition* $|(\mathsf{Ext}(X, U_d), U_d) - (U_m, U_d)| \leq \epsilon$*, where* U_m *and* U_d *are independent uniform strings on* m *and* d *bits respectively, and* $|\cdot|$ *stands for the statistical distance.*

Since their introduction, seeded randomness extractors have become fundamental objects in pseudorandomness, and have found numerous applications in derandomization, complexity theory, cryptography and many other areas in theoretical computer science. In addition, through a long line of research, we now have explicit constructions of seeded randomness extractors with almost optimal parameters (e.g., [GUV09]). However, the complexity of randomness extractors is still much less studied and understood. For example, while in general explicit constructions of randomness extractors can be computed in polynomial time of the input size, some of the known constructions are actually more explicit than that. These include for example extractors based on universal hashing [CW79], and Trevisan's extractor [Tre01], which can be computed by highly uniform constant-depth circuits of polynomial size with parity gates. Thus a main question one can ask is: can we do better and construct good randomness extractors with very low complexity?

This question is interesting not just by its own right, but also because such extractors, as building blocks, can be used to potentially reduce the complexity of other important objects. In this chapter we study this question and consider the parallel and local complexity of randomness extractors.

The parallel-$\mathcal{AC}^0$ model. The hierarchy of $\mathcal{NC}$ and $\mathcal{AC}$ circuits are standard

models for parallel computation. It is easy to see that the class of $\mathcal{NC}^0$ or even ℓ-local functions for small ℓ , which correspond to functions where each output bit depends on at most ℓ input bits (including both the weak source and the seed), cannot compute strong extractors (since one can just fix ℓ bits of the source). Thus, a natural relaxation is to consider the class $\mathcal{AC}^0$, which refers to the family of polynomial-size and constant-depth circuits with unbounded fan-in gates. Note that although we have strong lower bounds here for explicit functions, it is still not clear whether some important objects, such as randomness extractors and pseudorandom generators, can be computed in $\mathcal{AC}^0$ with good parameters. Thus the study of this question also helps us better understand the power of this class.

Viola [Vio05b] was the first to consider this question, and his result was generalized by Goldreich et al. [GVW15] to show that for strong seeded extractors, even extracting a single bit is impossible if $k < n/\mathsf{poly}(\log n)$. When $k \geq n/\mathsf{poly}(\log n)$, Goldreich et al. showed how to extract $\Omega(\log n)$ bits using $O(\log n)$ bits of seed, or more generally how to extract $m < k/2$ bits using $O(m)$ bits of seed. Note that the seed length is longer than the output length.[2] When the extractor does not need to be strong, they showed that extracting $r + \Omega(r)$ bits using r bits of seed is impossible if $k < n/\mathsf{poly}(\log n)$; while if $k \geq n/\mathsf{poly}(\log n)$ one can extract $(1+c)r$ bits for some constant $c > 0$, using r bits of seed. All the positive results here have error $1/\mathsf{poly}(n)$.

Therefore, a natural and main open problem left in [GVW15] is whether one can construct randomness extractors in $\mathcal{AC}^0$ with shorter seed and longer output. Specifically, [GVW15] asks if one can extract more than $\mathsf{poly}(\log n)r$ bits in $\mathcal{AC}^0$

[2] They also showed how to extract $\mathsf{poly}(\log n)$ bits using an $O(\log n)$ bit seed, but the error of the extractor becomes $1/\mathsf{poly}(\log n)$.

using a seed length $r = \Omega(\log n)$, when $k \geq n/\mathsf{poly}(\log n)$. In [GVW15] the authors conjectured that the answer is negative. Another open question is to see if one can achieve better error, e.g., negligible error instead of $1/\mathsf{poly}(n)$.

Goldreich et al. [GVW15] also studied deterministic extractors for bit-fixing sources, and most of their effort went into extractors for oblivious bit-fixing sources (although they also briefly studied non-oblivious bit-fixing sources). An (n,k)-oblivious bit-fixing source is a string of n bits such that some unknown k bits are uniform, while the other $n-k$ bits are fixed. Extractors for such sources are closely related to exposure-resilient cryptography [Can+00; KZ07]. In this case, a standard application of Håstad's switching lemma [Hås89] implies that it is impossible to construct extractors in $\mathcal{AC}^0$ for bit-fixing sources with min-entropy $k < n/\mathsf{poly}(\log n)$. The main result in [GVW15] is a theorem which shows the *existence* of deterministic extractors in $\mathcal{AC}^0$ for min-entropy $k \geq n/\mathsf{poly}(\log n)$ that output $k/\mathsf{poly}(\log n)$ bits with error $2^{-\mathsf{poly}(\log n)}$. We emphasize that this is an existential result, and [GVW15] did not give any explicit constructions of such extractors.

1.1.1 Our Results

As in [Vio05b; GVW15], in this chapter we obtain both negative results and positive results about randomness extraction in $\mathcal{AC}^0$. While the negative results in [Vio05b; GVW15] provide lower bounds on the entropy required for $\mathcal{AC}^0$ extractors, our negative results provide lower bounds on the error such extractors can achieve. We show that such extractors (both seeded extractors and deterministic extractors for bit-fixing sources) cannot achieve error better than $2^{-\mathsf{poly}(\log n)}$, even if the entropy of the sources is quite large. Specifically, we have

Theorem 1.1.2. *(General weak source) If* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *is a strong* $(k = n-1, \epsilon)$*-extractor that can be computed by* $\mathcal{AC}^0$ *circuits of depth* $\mathtt{dth}$ *and size* s*, then* $\epsilon = 2^{-(O(\log s))^{\mathtt{dth}-1}\log(n+d)}$.

(Bit-fixing source) There is a constant $c > 1$ *such that if* $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}^m$ *is a* (k,ϵ)*-extractor for oblivious bit-fixing sources with* $k = n - (c\log s)^{\mathtt{dth}-1}$*, that can be computed by* $\mathcal{AC}^0$ *circuits of depth* $\mathtt{dth}$ *and size* s*, then* $\epsilon = 2^{-(O(\log s))^{\mathtt{dth}-1}\log n}$. [3]

Thus, our results combined with the lower bounds on the entropy requirement in [Vio05b; GVW15] almost completely characterize the power of randomness extractors in $\mathcal{AC}^0$.

We now turn to our positive results. As our first contribution, we show that the authors' conjecture about seeded $\mathcal{AC}^0$ extractors in [GVW15] is false. We give explicit constructions of *strong* seeded extractors in $\mathcal{AC}^0$ with much better parameters. This in particular answers open problems 8.1 and 8.2 in [GVW15]. To start with, we have the following theorem.

Theorem 1.1.3. *For any constant* $c \in \mathbb{N}$*, any* $k = \Omega(n/\log^c n)$ *and any* $\epsilon = 1/\mathsf{poly}(n)$*, there exists an explicit construction of a strong* (k,ϵ)*-extractor* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *that can be computed by an* $\mathcal{AC}^0$ *circuit of depth* $c+10$*, where* $d = O(\log n)$ *,* $m = k^{\Omega(1)}$ *and the extractor family has locality* $O(\log^{c+5} n)$.

Note that the depth of the circuit is almost optimal, within an additive $O(1)$ factor of the lower bound given in [GVW15]. In addition, our construction is also

[3] This holds even if we allow Ext to have a uniform random seed, see Theorem 1.3.3.

a family with locality only $\mathsf{poly}(\log n)$. Note that the seed length $d = O(\log n)$ is (asymptotically) optimal, while the locality beats the one obtained in [BG13] (which is $O(n/m \log(m/\varepsilon) \log(n/m)) = n^{\Omega(1)}$) and is within a $\log^4 n$ factor to $O(n/k \log(n/\varepsilon))$.

Our result also improves that of De and Trevisan [DT09], even in the high min-entropy case, as our error can be any $1/\mathsf{poly}(n)$ instead of just $n^{-\alpha}$ for some constant $0 < \alpha < 1$. Moreover, our seed length remains $O(\log n)$ even for $k = n/\mathsf{poly}(\log n)$, while in this case the extractor in [DT09] has seed length $\mathsf{poly}(\log n)$.

Next, we can boost our construction to reduce the error and extract almost all the entropy. We have

Theorem 1.1.4. *For any constant $\gamma \in (0,1)$, $a, c \in \mathbb{N}$, any $k = \delta n = \Omega(n/\log^c n)$, $\epsilon = 1/2^{O(\log^a n)}$, there exists an explicit strong (k,ϵ)-extractor $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ in $\mathcal{AC}^0$ with depth $O(a+c+1)$ where $d = O((\log n + \frac{\log(n/\epsilon)\log(1/\epsilon)}{\log n})/\delta)$, $m = (1-\gamma)k$.*

As our second contribution, we give *explicit* deterministic extractors in $\mathcal{AC}^0$ for oblivious bit-fixing sources with entropy $k \geq n/\mathsf{poly}(\log n)$, which output $(1-\gamma)k$ bits with error $2^{-\mathsf{poly}(\log n)}$. This is in contrast to the non-explicit existential result in [GVW15]. Further, the output length and error of our extractor are almost optimal, while the output length in [GVW15] is only $k/\mathsf{poly}(\log n)$. Specifically, we have

Theorem 1.1.5. *For any constant $a, c \in \mathbb{N}$ and any constant $\gamma \in (0,1]$, there exists an explicit deterministic $(k = \Omega(n/\log^a n), \epsilon = 2^{-\log^c n})$-extractor $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}^{(1-\gamma)k}$ that can be computed by $\mathcal{AC}^0$ circuits of depth $O(a+c+1)$, for any (n,k)-bit-fixing source.*

1.1.2 Applications to pseudorandom generators in $\mathcal{AC}^0$

Like extractors, pseudorandom generators are also fundamental objects in the study of pseudorandomness, and constructing "more explicit" pseudorandom generators is another interesting question that has gained a lot of attention. A pseudorandom generator (or PRG for short) is an efficient deterministic function that maps a short random seed into a long output that looks uniform to a certain class of distinguishers.

Definition 1.1.6. *A function $G : \{0,1\}^n \to \{0,1\}^m$ is a pseudorandom generator for a class $\mathcal{C}$ of Boolean functions with error ε, if for every function $A \in \mathcal{C}$, we have that*

$$|\Pr[A(U_m) = 1] - \Pr[A(G(U_n)) = 1]| \leq \varepsilon.$$

Here we mainly consider two kinds of pseudorandom generators, namely cryptographic PRGs, which are necessarily based on computational assumptions; and unconditional PRGs, most notably PRGs for space bounded computation.

Standard cryptographic PRGs (i.e., PRGs that fool polynomial time computation or polynomial size circuits with negligible error) are usually based on one-way functions (e.g., [Hås+93]), and can be computed in polynomial time. However, more explicit PRGs have also been considered in the literature, for the purpose of constructing more efficient cryptographic protocols. Impagliazzo and Naor [IN96] showed how to construct such a PRG in $\mathcal{AC}^0$, which stretches n bits to $n + \log n$ bits. Their construction is based on the assumed intractability of the subset sum problem. On the other hand, Viola [Vio05a] showed that there is no black-box PRG construction with linear stretch in $\mathcal{AC}^0$ from one-way functions. Thus, to get such stretch one must use

non black-box constructions.

In [AIK06; AIK08], Applebaum et al. showed that the existence of cryptographic PRGs in $\mathcal{NC}^0$ with sub-linear stretch follows from a variety of standard assumptions, and they constructed a cryptographic PRG in $\mathcal{NC}^0$ with linear stretch based on a specific intractability assumption related to the hardness of decoding "sparsely generated" linear codes. In [App13], Applebaum further constructed PRG *collections* (i.e., a family of PRG functions) with linear stretch and polynomial stretch based on the assumption of one-wayness of a variant of the random local functions proposed by Goldreich [Gol11].

In the case of unconditional PRGs, for $d \geq 5$ Mossel et al. [MST06] constructed d-local PRGs with output length $n^{\Omega(d/2)}$ that fool all linear tests with error $2^{-n^{\frac{1}{2\sqrt{d}}}}$, which were used by Applebaum et al. [AIK06] to give a 3-local PRG with linear stretch that fools all linear tests. In the same paper, Applebaum et al. also gave a 3-local PRG with sub linear stretch that fools sublinear-space computation. Thus, it remains to see if we can construct better PRGs (cryptographic or unconditional) in $\mathcal{NC}^0$ or $\mathcal{AC}^0$ with better parameters.

Our PRGs We show that under reasonable computational assumptions, we can construct very good cryptographic PRGs in $\mathcal{AC}^0$ (e.g. with polynomial stretch and negligible error). In addition, we show that we can construct very good unconditional PRGs for space bounded computation in $\mathcal{AC}^0$ (e.g., with polynomial stretch).

We first give explicit cryptographic PRGs in $\mathcal{AC}^0$ based on the one-wayness of random local functions, the same assumption as used in [App13]. To state the assumption we first need the following definitions.

Definition 1.1.7 (Hypergraphs [App13])**.** *An (n, m, d) hypergraph is a graph over n vertices and m hyperedges each of cardinality d. For each hyperedge $S = (i_0, i_1, \ldots, i_{d-1})$, the indices $i_0, i_1, \ldots, i_{d-1}$ are ordered. The hyperedges of G are also ordered. Let G be denoted as $([n], S_0, S_1, \ldots, S_{m-1})$ where for $i = 0, 1, \ldots, m-1$, S_i is a hyperedge.*

Definition 1.1.8 (Goldreich's Random Local Function [Gol11])**.** *Given a predicate $Q : \{0,1\}^d \to \{0,1\}$ and an (n, m, d) hypergraph $G = ([n], S_0, \ldots, S_{m-1})$, the function $f_{G,Q} : \{0,1\}^n \to \{0,1\}^m$ is defined as follows: for input x, the ith output bit of $f_{G,Q}(x)$ is $f_{G,Q}(x)_i = Q(x_{S_i})$.*

For $m = m(n)$, the function collection $F_{Q,n,m} : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^m$ is defined via the mapping $(G, x) \to f_{G,Q}(x)$, where G is sampled randomly by the s bits and x is sampled randomly by the n bits.

For every $k \in \{0,1\}^s$, we also denote $F(k, \cdot)$ as $F_k(\cdot)$.

Definition 1.1.9 (One-wayness of a Collection of Functions)**.** *For $\epsilon = \epsilon(n) \in (0,1)$, a collection of functions $F : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^m$ is an ϵ-one-way function if for every efficient adversary A which outputs a list of $\mathsf{poly}(n)$ candidates and for sufficiently large n's, we have that*

$$\Pr_{k,x,y=F_k(x)}[\exists z \in A(k,y), z' \in F_k^{-1}(y), z = z'] < \epsilon,$$

where k and x are independent and uniform.

We now have the following theorem.

Theorem 1.1.10. *For any d-ary predicate Q, if the random local function $F_{Q,n,m}$ is δ-one-way for some constant $\delta \in (0,1)$, then we have the following results.*

1. *If there exists a constant* $\alpha > 0$ *such that* $m \geq (1+\alpha)n$*, then for any constant* $c > 1$*, there exists an explicit cryptographic PRG* $G : \{0,1\}^r \to \{0,1\}^t$ *in* $\mathcal{AC}^0$*, where* $t \geq cr$ *and the error is negligible*[4].

2. *If there exists a constant* $\alpha > 0$ *such that* $m \geq n^{1+\alpha}$*, then for any constant* $c > 1$ *there exists an explicit cryptographic PRG* $G : \{0,1\}^r \to \{0,1\}^t$ *in* $\mathcal{AC}^0$*, where* $t \geq r^c$ *and the error is negligible.*

As noted in [App13], there are several evidence supporting this assumption. In particular, current evidence is consistent with the existence of a δ-one-way random local function $F_{Q,n,m}$ with $m \geq n^{1+\alpha}$ for some constant $\alpha > 0$.

Compared to the constructions in [App13], our construction is in $\mathcal{AC}^0$ instead of $\mathcal{NC}^0$. However, our construction has the following advantages.

- We construct a standard PRG instead of a PRG collection, where the PRG collection is a family of functions and one needs to randomly choose one function before any application.

- The construction of a PRG with polynomial stretch in [App13] can only achieve polynomially small error, and for negligible error one needs to assume that the random local function cannot be inverted by any adversary with slightly super polynomial running time. Our construction, on the other hand, achieves negligible error while only assuming that the random local function cannot be inverted by any adversary that runs in polynomial time.

Next we give an explicit PRG in $\mathcal{AC}^0$ with polynomial stretch, that fools space

[4]The error $\epsilon : \mathbb{N} \to [0,1]$ is negligible if $\epsilon(n) = n^{-\omega(1)}$.

bounded computation. It is a straight forward application of our $\mathcal{AC}^0$-extractor to the Nisan-Zuckerman PRG [NZ96].

Theorem 1.1.11. *For every constant $c \in \mathbb{N}$ and every $m = m(s) = \mathsf{poly}(s)$, there is an explicit PRG $g : \{0,1\}^{r=O(s)} \to \{0,1\}^m$ in $\mathcal{AC}^0$, such that for any randomized algorithm A using space s,*

$$|\Pr[A(g(U_r)) = 1] - \Pr[A(U_m) = 1]| = \epsilon \leq 2^{-\Theta(\log^c s)},$$

where U_r is the uniform distribution of length r, U_m is the uniform distribution of length m.

Compared to the Nisan-Zuckerman PRG [NZ96], our PRG is in $\mathcal{AC}0$, which is more explicit. On the other hand, our error is $2^{-\Theta(\log^c s)}$ for any constant $c > 0$ instead of being exponentially small as in [NZ96]. It is a natural open problem to see if we can reduce the error to exponentially small. We note that this cannot be achieved by simply hoping to improve the extractor, since our negative result shows that seeded extractors in $\mathcal{AC}^0$ cannot achieve error better than $2^{-\mathsf{poly}(\log n)}$.

1.1.3 Overview of the Constructions and Techniques

Our negative results about the error of $\mathcal{AC}^0$ extractors follow by a simple application of Fourier analysis and the well known spectrum concentration theorem of $\mathcal{AC}^0$ functions [LMN93]. We present it in Section 1.3. We now briefly describe our positive results. We will extensively use the following two facts: the parity and inner product over $\mathsf{poly}(\log n)$ bits can be computed by $\mathcal{AC}^0$ circuits of size $\mathsf{poly}(n)$; in addition, any Boolean function on $O(\log n)$ bits can be computed by a depth-2 $\mathcal{AC}^0$ circuit of size $\mathsf{poly}(n)$.

Basic construction All our constructions are based on a basic construction of a strong extractor in $\mathcal{AC}^0$ for any $k \geq \frac{n}{\mathsf{poly}(\log n)}$ with seed length $d = O(\log n)$ and error $\varepsilon = n^{-\Omega(1)}$. This construction is a a modification of the Impagliazzo-Wigderson pseudorandom generator [IW97], interpreted as a randomness extractor in the general framework found by Trevisan [Tre01]. The IW-generator first takes a Boolean function on $\log n$ bits, applies a series of hardness amplifications to get another Boolean function on $O(\log n)$ bits, and then uses the Nisan-Wigderson generator [NW94] together with the new Boolean function. The hardness amplification consists of three steps: the first step, developed by Babai et al. [Bab+93], is to obtain a mild average-case hard function from a worst-case hard function; the second step involves a constant number of substeps, with each substep amplifying the hardness by using Impagliazzo's hard core set theorem [Imp95]; the third step, developed by Impagliazzo and Wigderson [IW97], uses a derandomized direct-product generator to obtain a function that can only be predicted with exponentially small advantage.

Trevisan [Tre01] showed that given an (n,k)-source X, if one regards the n bits of X as the truth table of the initial Boolean function on $\log n$ bits and applies the IW-generator, then by setting parameters appropriately one obtains an extractor. The reason is that for any $x \in \mathsf{supp}(X)$ that makes the output of the extractor fail a certain statistical test T, one can "reconstruct" x by showing that it can computed by a small size circuit, when viewing x as the truth table of the function with T gates. Thus the number of such bad elements $x \in \mathsf{supp}(X)$ is upper bounded by the total number of such circuits. This extractor works for any min-entropy $k \geq n^{\alpha}$.

However, this extractor itself is not in $\mathcal{AC}^0$ (which is not surprising since it can handle min-entropy $k \geq n^{\alpha}$). Thus, at least one of the steps in the construction of the

IW-generator/extractor is not in $\mathcal{AC}^0$. By carefully examining each step one can see that the only step not in $\mathcal{AC}^0$ is actually the first step of hardness amplification (This was also pointed out by [Vio05b]). Indeed, all the other steps of hardness amplification are essentially doing the same thing: obtaining a function f' on $O(\log n)$ bits from another function f on $O(\log n)$ bits, where the output of f' is obtained by taking the inner product over two $O(\log n)$ bit strings s and r. In addition, s is obtained directly from part of the input of f', while r is obtained by using the other part of the input of f' to generate $O(\log n)$ inputs to f and concatenating the outputs. All of these steps can be done in $\mathcal{AC}^0$, assuming f is in $\mathcal{AC}^0$ (note that f here depends on X).

We therefore modify the IW-generator by removing the first step of hardness amplification, and start with the second step of hardness amplification with the source X as the truth table of the initial Boolean function. Thus the initial function f can be computed by using the $\log n$ input bits to select a bit from X, which can be done in $\mathcal{AC}^0$. Therefore the final Boolean function f' can be computed in $\mathcal{AC}^0$. The last step of the construction, which applies the NW-generator, is just computing f' on several blocks of size $O(\log n)$, which certainly is in $\mathcal{AC}^0$. This gives our basic extractor in $\mathcal{AC}^0$.

The analysis is again similar to Trevisan's argument [Tre01]. However, since we have removed the first step of hardness amplification, now for any $x \in \mathsf{supp}(X)$ that makes the output of the extractor to fail a certain statistical test T, we cannot obtain a small circuit that *exactly computes* x. On the other hand, we can obtain a small circuit that can *approximate* x well, i.e., can compute x correctly on $1-\gamma$ fraction of inputs for some $\gamma = 1/\mathsf{poly}(\log n)$. We then argue that the total number of strings within relative distance γ to the outputs of the circuit is bounded, and therefore combining

the total number of possible circuits we can again get a bound on the number of such bad elements in $\mathsf{supp}(X)$. A careful analysis shows that our extractor works for any min-entropy $k \geq n/\mathsf{poly}(\log n)$. However, to keep the circuit size small we have to set the output length to be small enough, i.e., n^{α} and set the error to be large enough, i.e., $n^{-\beta}$.

Error reduction We now describe how we reduce the error of the extractor. We will borrow some techniques from the work of Raz et al. [RRV99], where the authors showed a general way to reduce the error of strong seeded extractors. However, the techniques in Raz et al. [RRV99] do not preserve the $\mathcal{AC}^0$ property, thus our techniques are significantly different from theirs. Nevertheless, our starting point is a lemma from [RRV99], which roughly says the following: given any strong seeded (k,ε)-extractor Ext with seed length d and output length m, then for any $x \in \{0,1\}^n$ there exists a set $G_x \subset \{0,1\}^d$ of density $1 - O(\varepsilon)$, such that if X is a source with entropy slightly larger than k, then the distribution $\mathsf{Ext}(X, G_X)$ is very close to having min-entropy $m - O(1)$. Here $\mathsf{Ext}(X, G_X)$ is the distribution obtained by first sampling x according to X, then sampling r uniformly in G_x and outputting $\mathsf{Ext}(x,r)$.

Giving this lemma, we can apply our basic $\mathcal{AC}^0$ extractor with error $\varepsilon = n^{-\beta}$ for some t times, each time with fresh random seed, and then concatenate the outputs. By the above lemma, the concatenation is roughly $(O(\varepsilon))^t$-close to a source such that one of the output has min-entropy $m - O(1)$ (i.e., a somewhere high min-entropy source). By choosing t to be a large enough constant the $(O(\varepsilon))^t$ can be smaller than any $1/\mathsf{poly}(n)$. We now describe how to extract from the somewhere high min-entropy source with error smaller than any $1/\mathsf{poly}(n)$, in $\mathcal{AC}^0$. This is where our construction differs significantly from [RRV99], as there one can simply apply a good extractor for

constant entropy rate.

Assume that we have an $\mathcal{AC}^0$ extractor Ext' that can extract from $(m, m-\sqrt{m})$-sources with error any $\varepsilon' = 1/\mathsf{poly}(n)$ and output length $m^{1/3}$. Then we can extract from the somewhere high min-entropy source as follows. We use Ext' to extract from each row of the source with fresh random seed, and then compute the XOR of the outputs. We claim the output is $(2^{-m^{\Omega(1)}} + \varepsilon')$-close to uniform. To see this, assume without loss of generality that the i'th row has min-entropy $m - O(1)$. We can now fix the outputs of all the other rows, which has a total size of $tm^{1/3} \ll \sqrt{m}$ as long as t is small. Thus, even after the fixing, with probability $1 - 2^{-m^{\Omega(1)}}$, we have that the i'th row has min-entropy at least $m - \sqrt{m}$. By applying Ext' we know that the XOR of the outputs is close to uniform.

What remains is the extractor Ext'. To construct it we divide the source with length m sequentially into $m^{1/3}$ blocks of length $m^{2/3}$. Since the source has min-entropy $m - \sqrt{m}$, this forms a block source such that each block roughly has min-entropy at least $m^{2/3} - \sqrt{m}$ conditioned on the fixing of all previous ones. We can now take a strong extractor Ext'' in $\mathcal{AC}^0$ with seed length $O(\log n)$ and use the same seed to extract from all the blocks, and concatenate the outputs. It suffices to have this extractor output one bit for each block. Such $\mathcal{AC}^0$ extractors are easy to construct since each block has high min-entropy rate (i.e., $1 - o(1)$). For example, we can use the extractors given by Goldreich et al. [GVW15].

It is straightforward to check that our construction is in $\mathcal{AC}^0$, as long as the final step of computing the XOR of t outputs can be done in $\mathcal{AC}^0$. For error $1/\mathsf{poly}(n)$, it suffices to take t to be a constant and the whole construction is in $\mathcal{AC}^0$, with seed length $O(\log n)$. We can even take t to be $\mathsf{poly}(\log n)$, which will give us error

$2^{-\mathsf{poly}(\log n)}$.

Increasing output length The error reduction step reduces the output length from m to $m^{1/3}$, which is still $n^{\Omega(1)}$. We can increase the output length by using a standard boosting technique as that developed by Nisan and Zuckerman [NZ96; Zuc97]. Specifically, we first use random bits to sample several blocks from the source, using a sampler in $\mathcal{AC}^0$. We then apply our $\mathcal{AC}^0$ extractor on the blocks backwards, and use the output of one block as the seed to extract from the previous block. When doing this we divide the seed into blocks each with the same length as the seed of the $\mathcal{AC}^0$ extractor, apply the $\mathcal{AC}^0$ extractor using each block as the seed, and then concatenate the outputs. This way each time the output will increase by a factor of $n^{\Omega(1)}$. Thus after a constant number of times it will become say $\Omega(k)$. Since each step is computable in $\mathcal{AC}^0$, the whole construction is still in $\mathcal{AC}^0$.

Explicit $\mathcal{AC}^0$ extractors for bit-fixing source Our explicit $\mathcal{AC}^0$ extractors for (oblivious) bit-fixing sources follow the high-level idea in [GVW15]. Specifically, we first reduce the oblivious bit-fixing source to a non-oblivious bit-fixing source, and then apply an extractor for non-oblivious bit-fixing sources. This approach is natural in the sense that the best known extractors for oblivious bit-fixing sources (e.g., parity or [KZ07]) can both work for small entropy and achieve very small error. Thus by the negative results in [GVW15] and our result, none of these can be in $\mathcal{AC}^0$. However, extractors for non-oblivious bit-fixing sources are equivalent to resilient functions, and there are well known resilient functions in $\mathcal{AC}^0$ such as the Ajtai-Linial function [AL93].

The construction in [GVW15] is not explicit, but only existential for two reasons.

First, at that time the Ajtai-Linial function is a random function, and there was no explicit construction matching it. Second, the conversion from oblivious-bit fixing source to non-oblivious bit-fixing source in [GVW15] is to multiply the source by a random matrix, for which the authors of [GVW15] showed its existence but were not able to give an explicit construction. Now, the first obstacle is solved by recent explicit constructions of resilient functions in $\mathcal{AC}^0$ that essentially match the Ajtai-Linial function ([CZ16; Mek15; Li16]). Here we use the extractor in [Li16] that can output many bits. For the second obstacle, we notice that the extractors for non-oblivious bit-fixing sources in [CZ16; Li16] do not need the uniform bits to be independent, but rather only require $\mathsf{poly}(\log N)$-wise independence if N is the length of the source.

By exploiting this property, we can give an explicit construction of the matrix used to transfrom the original oblivious bit-fixing source. Our construction is natural and simpler than that in [GVW15], in the sense that it is a matrix over F_2 while the matrix in [GVW15] uses fields of larger size. Specifically, we will take a seeded extractor and view it as a bipartite graph with $N = n^{O(1)}$ vertices on the left, n vertices on the right and left degree $d = \mathsf{poly}(\log N) = \mathsf{poly}(\log n)$. We identify the right vertices with the n bits of the bit-fixing source, and for each left vertex we obtain a bit which is the parity of its neighbors. The new non-oblivious bit-fixing source is the N bit source obtained by concatenating the left bits.

Now suppose the original source has entropy $k = \delta n$ for some $\delta \geq 1/\mathsf{poly}(\log n)$, and let T denote the unfixed bits. A standard property of the seeded extractor implies that most of the left vertices have a good fraction of neighbors in T (i.e., an extractor is a good sampler), so that each left bit obtained from these vertices is uniform. Next we would like to argue that they are $\mathsf{poly}(\log N)$-wise independent. For this we

require the seeded extractor to have a stronger property: that it is a *design extractor* as defined by Li [Li12]. Besides being an extractor itself, a design extractor requires that any pair of left vertices have a small intersection of neighbors. Assuming this property, it is easy to show that if we take any small subset S of the "good" left vertices, then there is a bit in T that is only connected to a single vertex in S (i.e., a unique neighbor). Thus the XOR of any small enough subset of the "good" left bits is uniform, which indicates that they are some t-wise independent. Several explicit constructions of design extractors were given in [Li12], and for our applications it suffices to use a simple greedy construction. By adjusting the parameters, we can ensure that $t = \mathsf{poly}(\log N)$ which is enough for applying the extractor in [Li16]. In addition, the degree $d = \mathsf{poly}(\log N)$ so the parity of d bits can be computed in $\mathcal{AC}^0$.

Once we have the basic extractor, we can use the same techniques as in [GVW15] to reduce the error, and use the techniques by Gabizon et al.[GRS04] to increase the output length (this is also done in [GVW15]). Note that the techniques in [GRS04] require a seeded extractor. In order for the whole construction to be in $\mathcal{AC}^0$, we use our previously constructed seeded extractor in $\mathcal{AC}^0$ which can output $(1-\gamma)k$ bits. Thus we obtain almost optimal explicit $\mathcal{AC}^0$ extractors for oblivious bit-fixing sources. In contrast, the seeded extractor used in [GVW15] only outputs $k/\mathsf{poly}(\log n)$ bits, and thus their (non-explicit) $\mathcal{AC}^0$ extractor for oblivious bit-fixing sources also only outputs $k/\mathsf{poly}(\log n)$ bits.

Applications to pseudorandom generators For cryptographic pseudorandom generators, we mainly adapt the approach of Applebaum [App13], to the $\mathcal{AC}^0$ setting. The construction of cryptographic pseudorandom generator *families* in [App13] is based on random local functions. Specifically, given a random bipartite graph with

n left vertices, m right vertices and right degree d (think of d as a constant), and a suitable predicate P on d bits, Applebaum showed that based on a conjecture on random local one-way functions, the m output bits obtained by applying P to the m subsets of input bits corresponding to the hyper edges give a distribution with high pseudo Shannon entropy. He then showed how to boost the output to have high pseudo min-entropy by concatenating several independent copies. At this point he used an extractor in $\mathcal{NC}^0$ to turn the output into a pseudorandom string.

However, an extractor in $\mathcal{NC}^0$ needs to have a large seed length (i.e., $\Omega(n)$), thus the $\mathcal{NC}^0$ PRG constructed using this approach only achieves linear stretch. Another issue is that the $\mathcal{NC}^0$ PRG is actually a collection of functions rather than a single function, because the random bits used to sample the bipartite graph is larger than the output length, and is treated as a public index to the collection of functions.

Here, by replacing the extractor with our $\mathcal{AC}^0$ extractor we can achieve a polynomial stretch PRG (based on appropriate assumptions as in [App13]), although now the PRG is in $\mathcal{AC}^0$ instead of $\mathcal{NC}^0$. In addition, we can get a single PRG instead of a collection of PRG functions, by including the random bits used to sample the bipartite graph as part of the seed. Since in the graph each right vertex only has a constant number d of neighbors, the sampling uses $md \log n$ bits and can be done in $\mathcal{AC}^0$. To ensure that the PRG has a stretch, we take the sampled graph G and apply the *same* graph to several independent copies of n bit input strings. We show that we can still use the method in [App13] to argue that this gives a a distribution with high pseudo Shannon entropy. We then use the same method as in [App13] to turn it into a distribution with high pseudo min-entropy, and finally we apply our $\mathcal{AC}^0$ extractor. This way we ensure that the $md \log n$ bits used to sample the graph G are “absorbed"

by the stretch of the PRG, and thus we get a standard PRG instead of a collection of PRG functions.

For PRGs for space bounded computation, we simply adapt the PRG by Nisan and Zuckerman [NZ96], which stretches $O(S)$ random bits to any $\mathsf{poly}(S)$ bits that fool space S computation. We now replace the seeded extractor used there by our $\mathcal{AC}^0$ extractor. Notice that the Nisan-Zuckerman PRG simply applies the seeded extractor iteratively for a constant number of times, so the whole construction is still in $\mathcal{AC}^0$.

1.1.4 Open Problems

Our work leaves many natural open problems. First, in terms of the seed length and output length, our $\mathcal{AC}^0$ extractor is only optimal when $k = \Omega(n)$. Is it possible to simultaneously achieve optimal seed length and output length when $k = n/\mathsf{poly}(\log n)$? Second, can we construct good $\mathcal{AC}^0$ extractors for other classes of sources, such as independent sources and affine sources?

Turning to strong extractor families with small locality, again the parameters of our constructions do not match the parameters of optimal seeded extractors. In particular, our seed length is still $O(k)$ when the min-entropy k is small. Can we reduce the seed length further? We note that using our analysis together with the IW-generator/extractor, one can get something meaningful (i.e., a strong extractor family with a relatively short seed and small locality) even when $k = n^{\alpha}$ for some $\alpha > 1/2$. But it's unclear how to get below this entropy.

For pseudorandom generators in $\mathcal{AC}^0$, there are also many interesting open problems left. For example, can we construct better cryptographic PRGs, or use weaker

computational assumptions? In particular, it would be nice to construct a cryptographic PRG with polynomial stretch based on the one-wayness of a random local function with $m = (1+\alpha)n$ instead of $m = n^{1+\alpha}$ as in our current construction. For space bounded computation, is it possible to match the exponentially small error of the Nisan-Zukerman PRG? Taking one step further, is it possible to construct PRGs in $\mathcal{AC}^0$ for space bounded computation, with stretch matching the PRGs of Nisan [Nis92] and Impagliazzo-Nisan-Wigderson [INW94]?

1.1.5 Chapter Organization

The rest of the chapter is organized as follows. In Section 1.2 we review some basic definitions and the relevant background. In Section 1.3, we give the lower bounds on errors of $\mathcal{AC}^0$ extractors for general weak sources and bit-fixing sources. In Section 1.4 we describe our construction of a basic extractor in $\mathcal{AC}^0$, and with small locality. In section 1.5 we describe the error reduction techniques for $\mathcal{AC}^0$ extractors. In Section 1.6 we show how to increase the output length for our $\mathcal{AC}^0$ extractors. In section 1.7 we construct $\mathcal{AC}^0$ extractors for bit-fixing sources. In Section 1.8 we present several applications, i.e., constructing pseudorandom generators in $\mathcal{AC}^0$.

1.2 Preliminaries

For any $i \in \mathbb{N}$, we use $\langle i \rangle$ to denote the string which is the binary representation of i. Let $\langle \cdot, \cdot \rangle$ denote the inner product of two binary strings having the same length. Let $|\cdot|$ denote the length of the input string. Let $w(\cdot)$ denote the weight of the input binary string. For any strings x_1 and x_2, let $x_1 \circ x_2$ denote the concatenation of x_1 and x_2. For any strings $x_1, x_2, \ldots, x_t$, let $\bigcirc_{i=1}^{t} x_i$ denote $x_1 \circ x_2 \circ \cdots \circ x_t$.

Let $\mathsf{supp}(\cdot)$ denote the support of the input random variable.

Definition 1.2.1 (Weak Random Source, Block Source)**.** *The min-entropy of a random variable X is*

$$H_\infty(X) = \min_{x \in \mathsf{supp}(X)} \{-\log \Pr(X = x)\}.$$

We say a random variable X is an (n,k)-source if the length of X is n and $H_\infty(X) \geq k$. We say $X = \bigcirc_{i=1}^{m} X_i$ is an $((n_1,k_1),(n_2,k_2),\ldots,(n_m,k_m))$-block source if $\forall i \in [m]$, $\forall x \in \mathsf{supp}(\bigcirc_{j=1}^{i-1} X_j)$, $X_i|_{\bigcirc_{j=1}^{i-1} X_j = x}$ is an (n_i,k_i)-source.

For simplicity, if $n_1, n_2, \cdots, n_m$ are clear from the context, then we simply say that the block source X is a $(k_1,k_2,\ldots,k_m)$-block source.

We say an (n,k)-source X is a flat (n,k)-source if $\forall a \in \mathsf{supp}(X)$, $\Pr[X = a] = 2^{-k}$. In this paper, X is usually a random binary string with finite length. So $\mathsf{supp}(X)$ includes all the binary strings of that length such that $\forall x \in \mathsf{supp}(X), \Pr[X = x] > 0$.

Bit-fixing source is a special kind of weak source. In this paper we also consider deterministic extractors for bit-fixing source.

Definition 1.2.2 (Non-oblivious Bit-Fixing Sources)**.** *A source X on $\{0,1\}^n$ is a (q,t,γ)-non-oblivious bit-fixing source (in short, NOBF source) if there exists a subset $Q \subseteq [n]$ of size at most q and a sequence of functions $f_1, f_2, \ldots, f_n : \{0,1\}^{|I|} \to \{0,1\}$ such that the joint distribution of the bits indexed by $\bar{Q} = [n] \backslash Q$ (denoted by $X_{\bar{Q}}$) is (t,γ)-wise independent (γ-close to a t-wise independent source) and $X_i = f_i(X_{\bar{Q}})$ for every $i \in Q$.*

Bit-fixing sources are special non-oblivious bit-fixing sources. An (n,t)-bit-fixing source is defined to be an $(n-t,t,0)$-non-oblivious bit-fixing source.

We use U to denote the uniform distribution. In the following, we do not always claim the length of U, but its length can be figured out from the context.

Definition 1.2.3 (Statistical Distance)**.** *The statistical distance between two random variables X and Y, where $|X| = |Y|$, is $\mathsf{SD}(X,Y)$ which is defined as follows.*

$$\mathsf{SD}(X,Y) = 1/2 \sum_{a \in \{0,1\}^{|X|}} |\Pr[X=a] - \Pr[Y=a]|$$

Lemma 1.2.4 (Properties of Statistical Distance [AB09])**.** *Statistical distance has the following properties.*

1. *(Triangle Inequality) For any random variables X, Y, Z, such that $|X| = |Y| = |Z|$, we have*
$$\mathsf{SD}(X,Y) \leq \mathsf{SD}(X,Z) + \mathsf{SD}(Y,Z).$$
2. *For any $n, m \in \mathbb{N}^+$, any deterministic function $f : \{0,1\}^n \to \{0,1\}^m$ and any random variables X, Y over $\{0,1\}^n$, $\mathsf{SD}(f(X), f(Y)) \leq \mathsf{SD}(X,Y)$.*

Definition 1.2.5 (Extractor)**.** *A (k,ϵ)-extractor is a function $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with the following property. For every (n,k)-source X, the distribution $\mathsf{Ext}(X,U)$ is within statistical distance ϵ from uniform distributions over $\{0,1\}^m$.*

A strong (k,ϵ)-extractor is a function $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with the following property. For every (n,k)-source X, the distribution $U \circ \mathsf{Ext}(X,U)$ is within statistical distance ϵ from uniform distributions over $\{0,1\}^{d+m}$. The entropy loss of the extractor is $k-m$.

The existence of extractors can be proved using the probabilistic method. The result is stated as follows.

Theorem 1.2.6 ([Vad12]). *For any $n, k \in \mathbb{N}$ and $\epsilon > 0$, there exists a strong (k,ϵ)-extractor $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ such that $d = \log(n-k) + 2\log(1/\epsilon) + O(1), m = k - 2\log(1/\epsilon) + O(1)$.*

In addition, researchers have found explicit extractors with almost optimal parameters, for example we have the following theorem.

Theorem 1.2.7 ([GUV09]). *For every constant $\alpha > 0$, every $n, k \in \mathbb{N}$ and $\epsilon > 0$, there exist an explicit construction of strong (k,ϵ)-extractor $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $d = O(\log \frac{n}{\epsilon}), m \geq (1-\alpha)k$.*

We also use the following version of Trevisan's extractor [Tre01].

Theorem 1.2.8 (Trevisan's Extractor [Tre01]). *For any constant $\gamma \in (0,1]$, let $k = n^\gamma$. For any $\epsilon \in (0, 2^{-k/12})$, there exists an explicit construction of (k,ϵ)-extractor $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ such that $d = O((\log n/\epsilon)^2/\log n)$, $m \in [36, k/2)$.*

For block sources, randomness extraction can be done in parallel, using the same seed for each block.

Lemma 1.2.9 (Block Source Extraction). *For any $t \in \mathbb{N}^+$, let $X = \bigcirc_{i=1}^t X_i$ be any $(k_1, k_2, \ldots, k_t)$-block source where for each $i \in [t]$, $|X_i| = n_i$. For every $i \in [t]$, let $\mathsf{Ext}_i : \{0,1\}^{n_i} \times \{0,1\}^d \to \{0,1\}^{m_i}$ be a strong (k_i, ϵ_i)-extractor. Then the distribution $R \circ \mathsf{Ext}_1(X_1, R) \circ \mathsf{Ext}_2(X_2, R) \circ \ldots \circ \mathsf{Ext}_t(X_t, R)$ is $\sum_{i\in[t]} \epsilon_i$-close to uniform, where R is uniformly sampled from $\{0,1\}^d$, and independent of X.*

Proof. We use induction. If the source has only 1 block, then the statement is true by the definition of strong extractors.

Assume for $(t-1)$ blocks, the statement is true. We view $\mathsf{Ext}_1(X_1,R)\circ\mathsf{Ext}_2(X_2,R)\circ\ldots\circ\mathsf{Ext}_t(X_t,R)$ as $Y\circ\mathsf{Ext}_t(X_t,R)$. Here $Y=\mathsf{Ext}_1(X_1,R)\circ\mathsf{Ext}_2(X_2,R)\circ\ldots\circ\mathsf{Ext}_{t-1}(X_{t-1},R)$. Let U_1,U_2 be two independent uniform distributions, where $|U_1|=|Y|=m$ and $|U_2|=m_t$. Then

$$\begin{aligned}&\mathsf{SD}(R\circ Y\circ\mathsf{Ext}_t(X_t,R),R\circ U_1\circ U_2)\\ \leq&\mathsf{SD}(R\circ Y\circ\mathsf{Ext}_t(X_t,R),R\circ U_1\circ Z)+\mathsf{SD}(R\circ U_1\circ Z,R\circ U_1\circ U_2).\end{aligned}\tag{1.1}$$

Here Z is the random variable such that $\forall r\in\{0,1\}^d,\forall y\in\{0,1\}^m$, $Z|_{R=r,U_1=y}$ has the same distribution as $\mathsf{Ext}_t(X_t,R)|_{R=r,Y=y}$.

First we give the upper bound of $\mathsf{SD}(R\circ Y\circ\mathsf{Ext}_t(X_t,R),R\circ U_1\circ Z)$.

$$\mathsf{SD}(R \circ Y \circ \mathsf{Ext}_t(X_t, R), R \circ U_1 \circ Z) \tag{1.2}$$

$$= \frac{1}{2} \sum_{r \in \{0,1\}^d} \sum_{y \in \{0,1\}^m} \sum_{z \in \{0,1\}^{m_t}} |\Pr[R = r] \Pr[Y = y|_{R=r}] \Pr[\mathsf{Ext}_t(X_t, R) = z|_{R=r, Y=y}]$$

$$- \Pr[R = r] \Pr[U_1 = y] \Pr[Z = z|_{R=r, U_1=y}]|$$

$$= \frac{1}{2} \sum_{r \in \{0,1\}^d} \sum_{y \in \{0,1\}^m} \sum_{z \in \{0,1\}^{m_t}} \Pr[R = r] \Pr[Z = z|_{R=r, U_1=y}] |\Pr[Y = y|_{R=r}] - \Pr[U_1 = y]|$$

$$= \frac{1}{2} \sum_{r \in \{0,1\}^d} \sum_{y \in \{0,1\}^m} \Pr[R = r] |\Pr[Y = y|_{R=r}] - \Pr[U_1 = y]| \sum_{z \in \{0,1\}^{m_t}} \Pr[Z = z|_{R=r, U_1=y}]$$

$$= \frac{1}{2} \sum_{r \in \{0,1\}^d} \sum_{y \in \{0,1\}^m} \Pr[R = r] |\Pr[Y = y|_{R=r}] - \Pr[U_1 = y]|$$

$$= \mathsf{SD}(R \circ Y, R \circ U)$$

$$\leq \sum_{i=1}^{t-1} \epsilon_i.$$

Next we give the upper bound of $\mathsf{SD}(R \circ U_1 \circ Z, R \circ U_1 \circ U_2)$.

$$\mathsf{SD}(R \circ U_1 \circ Z, R \circ U_1 \circ U_2) \tag{1.3}$$

$$= \frac{1}{2} \sum_{r \in \{0,1\}^r} \sum_{u \in \{0,1\}^m} \sum_{z \in \{0,1\}^{m_t}} |\Pr[R = r] \Pr[U_1 = u] \Pr[Z = z|_{R=r, U_1=u}]$$

$$- \Pr[R = r] \Pr[U_1 = u] \Pr[U_2 = z]|$$

$$= \frac{1}{2} \sum_{r \in \{0,1\}^r} \sum_{u \in \{0,1\}^m} \sum_{z \in \{0,1\}^{m_t}} \Pr[R = r] \Pr[U_1 = u] |\Pr[Z = z|_{R=r, U_1=u}] - \Pr[U_2 = z]|$$

$$= \frac{1}{2} \sum_{u \in \{0,1\}^m} \sum_{r \in \{0,1\}^r} \sum_{z \in \{0,1\}^{m_t}} \Pr[R = r] \Pr[U_1 = u] |\Pr[Z = z|_{R=r, U_1=u}] - \Pr[U_2 = z]|$$

$$= \frac{1}{2} \sum_{u \in \{0,1\}^m} \Pr[U_1 = u] \sum_{r \in \{0,1\}^r} \sum_{z \in \{0,1\}^{m_t}} \Pr[R = r] |\Pr[Z = z|_{R=r, U_1=u}] - \Pr[U_2 = z]|$$

$$= \frac{1}{2} \sum_{u \in \{0,1\}^m} \Pr[U_1 = u] \sum_{r \in \{0,1\}^r} \sum_{z \in \{0,1\}^{m_t}} \Pr[R = r] |\Pr[\mathsf{Ext}_t(X_t, R) = z|_{R=r, Y=u}] - \Pr[U_2 = z]|$$

$$= \sum_{u \in \{0,1\}^m} \Pr[U_1 = u] \mathsf{SD}(R \circ \mathsf{Ext}_t(X_t, R)|_{Y=u}, R \circ U_2)$$

$$\leq \sum_{u \in \{0,1\}^m} \Pr[U_1 = u] \epsilon_t$$

$$= \epsilon_t.$$

So $\mathsf{SD}(R \circ Y \circ \mathsf{Ext}_t(X_t, R), R \circ U_1 \circ U_2) \leq \sum_{i=1}^{t} \epsilon_t$. This proves the lemma. □

For any circuit C, the size of C is denoted as $\mathsf{size}(C)$. The depth of C is denoted as $\mathsf{depth}(C)$.

Definition 1.2.10 ($\mathcal{AC}^0$)**.** *$\mathcal{AC}^0$ is the complexity class which consists of all families of circuits having constant depth and polynomial size. The gates in those circuits are* NOT *gates,* AND *gates and* OR *gates where* AND *gates and* OR *gates have unbounded fan-in.*

Lemma 1.2.11. *The following are some well known properties of $\mathcal{AC}^0$ circuits. For any $n \in \mathbb{N}$,*

1. *([AB09] forklore) any boolean function $f : \{0,1\}^{l=\Theta(\log n)} \to \{0,1\}$ can be computed by an $\mathcal{AC}^0$ circuit of size $\mathsf{poly}(n)$ and depth 2. In fact, it can be represented by either a CNF or a DNF.*

2. *([Gol+07]) for every $c \in \mathbb{N}$, every integer $l = \Theta(\log^c n)$, if the function $f_l : \{0,1\}^l \to \{0,1\}$ can be computed by circuits of depth $O(\log l)$ and size $\mathsf{poly}(l)$, then it can be computed by $\mathcal{AC}^0$ (in n) circuits of depth $c+1$.*

Proof. For the first assertion, for an input string $u \in \{0,1\}^l$,

$$f(u) = \bigvee_{j=0}^{2^l-1} (I_{u=\langle j \rangle} \wedge f(\langle j \rangle)) = \bigwedge_{j=0}^{2^l-1} (I_{u \neq \langle j \rangle} \vee f(\langle j \rangle)).$$

Here I_e is the indicator function such that $I_e = 1$ if e is true and $I_e = 0$ otherwise. We know that $I_{u=\langle j \rangle}$ can be represented as a boolean formula with only AND and NOT gates, checking whether $u = \langle j \rangle$ bit by bit. Similarly $I_{u \neq \langle j \rangle}$ can be represented as a boolean formula with only OR and NOT gates by taking the negation of $I_{u=\langle j \rangle}$. So the computation of obtaining $f(u)$ can be represented by a CNF/DNF. Thus it can be realized by a circuit of depth 2 by merging the gates of adjacent levels.

Next we prove the second assertion.

As there exists an $\mathcal{NC}^1$-complete problem which is downward self-reducible [Gol+07], f_l can be reduced to ($\mathcal{AC}^0$ reduction) to $f_{l'}$ where $l' = l^\alpha$, for any $\alpha \in (0,1)$. Once we let $l' = l^\alpha = O(\log n)$, f can be computed in $\mathcal{AC}^0$ (in n). The circuit depth is $c+1$, as $\mathcal{AC}^0$ reduction has depth c and $f_{l'}$ can be realized by CNF/DNFs.

□

Definition 1.2.12. *A boolean function $f : \{0,1\}^l \to \{0,1\}$ is δ-hard on uniform distributions for circuit size g, if for any circuit C with at most g gates ($\mathsf{size}(C) \leq g$), we have $\Pr_{x \leftarrow U}[C(x) = f(x)] < 1 - \delta$.*

Definition 1.2.13 (Graphs)**.** *Let $G = (V, E)$ be a graph. Let A be the adjacency matrix of G. Let $\lambda(G)$ be the second largest eigenvalue of A. We say G is d-regular, if the degree of G is d. When G is clear in the context, we simply denote $\lambda(G)$ as λ.*

1.3 Lower Bound for Error Parameters of $\mathcal{AC}^0$ Extractors

Here we show a lower bound on the error of strong $\mathcal{AC}^0$ seeded extractors. Our conclusion is mainly based on the well known LMN theorem deduced by Fourier analysis, given by Linial, Mansour, and Nisan [LMN93].

Let the Fourier expansion of a function $f : \{-1,1\}^n \to \{-1,1\}$ be $f(x) = \sum_{S \subseteq [n]} \hat{f}_S \chi_S(x)$, where $\chi_S(x) = \prod_{i=1}^n x_i$. For any $f, g : \{-1,1\}^n \to \{-1,1\}$, $\langle f, g \rangle = \frac{1}{2^n} \sum_{x \in \{-1,1\}^n} f(x)g(x)$.

Theorem 1.3.1 (LMN Theorem [LMN93] [O'D14])**.** *Let $f : \{-1,1\}^n \to \{-1,1\}$ be computable by $\mathcal{AC}^0$ circuits of size $s > 1$ and depth $\mathtt{dth}$. Let $\epsilon \in (0, 1/2]$. There*

exists $t = O(\log(s/\epsilon))^{\mathtt{dth}-1} \cdot \log(1/\epsilon)$ *s.t.*

$$\sum_{S\subseteq[n],|S|>t} \hat{f}_S^2 \leq \epsilon.$$

Our first lower bound is as the follows.

Theorem 1.3.2. *If* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *is a strong* $(k = n-1, \epsilon)$*-extractor that can be computed by* $\mathcal{AC}^0$ *circuits of depth* $\mathtt{dth}$ *and size* s*, then* $\epsilon = 2^{-(O(\log s))^{\mathtt{dth}-1}\log(n+d)}$.

Proof. Without loss of generality, let $m = 1$. Let's transform the function space of Ext to $\{-1,1\}^{n+d} \to \{-1,1\}$, achieving function f. Let $\epsilon_0 = 1/2$. By Theorem 1.3.1, there exists $t = O(\log(s/\epsilon_0))^{\mathtt{dth}-1} \cdot \log(1/\epsilon_0) = O(\log s)^{\mathtt{dth}-1}$ s.t.

$$\sum_{S\subseteq[n+d],|S|\leq t} \hat{f}_S^2 > 1 - \epsilon_0 = 1/2.$$

Fix an $S = S_1 \cup \{i+n \mid i \in S_2\}$ with $|S| \leq t$, where $S_1 \subseteq [n], S_2 \subseteq \{1,2,\ldots,d\}$. We know that

$$\hat{f}_S = \langle f, \chi_S \rangle = 1 - 2\Pr_u[f(u) \neq \chi_S(u)]$$

where u is uniformly drawn from $\{-1,1\}^{n+d}$.

For $a \in \{-1,1\}$, let X_a be the uniform distribution over $\{-1,1\}^n$ conditioned on $\prod_{i\in S_1} X_i = a$. Also for $b \in \{-1,1\}$, let R_b be the uniform distribution over $\{-1,1\}^d$ conditioned on $\prod_{i\in S_2} R_i = b$. So $\chi_S(x \circ r) = ab$ for $x \in \mathsf{supp}(X_a), r \in \mathsf{supp}(R_b)$. For special situations, saying $S_1 = \emptyset$ (or $S_2 = \emptyset$), let X_a (or R_b) be uniform.

As Ext is a strong (k,ϵ)-extractor, R_b only blows up the error by 2. Also note that

by definition, X_a has entropy $n-1$. So

$$\mathsf{dist}(f(X_a \circ R_b), U) \leq 2\epsilon,$$

where U is uniform over $\{-1,1\}$.

So

$$\forall a, b \in \{-1,1\}, |\Pr[f(X_a \circ R_b) \neq ab] - 1/2| \leq 2\epsilon.$$

Thus

$$\begin{aligned} |\Pr_u[f(u) \neq \chi_S(u)] - 1/2| &= |\sum_{a\in\{-1,1\}} \sum_{b\in\{-1,1\}} \frac{1}{4}(\Pr[f(X_a \circ R_b) \neq ab] - 1/2)| \\ &\leq \sum_{a\in\{-1,1\}} \sum_{b\in\{-1,1\}} \frac{1}{4}|\Pr[f(X_a \circ R_b) \neq ab] - 1/2| \\ &\leq 2\epsilon. \end{aligned} \tag{1.4}$$

Hence

$$|\hat{f}_S| = |1 - 2\Pr_u[f(u) \neq \chi_S(u)]| = 2|\Pr_u[f(u) \neq \chi_S(u)] - 1/2| \leq 4\epsilon.$$

As a result,

$$1/2 \leq \sum_{S\subseteq[n+d],|S|\leq t} \hat{f}_S^2 \leq \sum_{i=0}^{t} \binom{n+d}{i} (4\epsilon)^2.$$

So

$$\epsilon \geq \sqrt{\frac{1}{32\sum_{i=0}^{t}\binom{n+d}{i}}}.$$

As $\sum_{i=0}^{t}\binom{n+d}{i} \leq (\frac{e(n+d)}{t})^t = 2^{O(t\log(n+d))} = 2^{O(\log s)^{\mathtt{dth}-1}\log(n+d)}$, $\epsilon = 2^{-O(\log s)^{\mathtt{dth}-1}\log(n+d)}$.

□

We also consider extractors for bit-fixing sources and give the following negative result on the error.

Theorem 1.3.3. *There is a constant $c > 1$ such that if* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *is a strong (k,ϵ)-extractor for oblivious bit-fixing sources with $k = n - (c \log s)^{\mathtt{dth}-1}$, that can be computed by $\mathcal{AC}^0$ circuits of depth* $\mathtt{dth}$ *and size s, then $\epsilon = 2^{-(O(\log s))^{\mathtt{dth}-1} \log(n+d)}$.*

The proof is slightly different from that of theorem 1.3.2.

Proof. Let $m = 1$ and also transform the function space of Ext to $\{-1,1\}^{n+d} \to \{-1,1\}$, achieving function f. Let $\epsilon_0 = 1/2$. By Theorem 1.3.1, there exists $t = O(\log(s/\epsilon_0))^{\mathtt{dth}-1} \cdot \log(1/\epsilon_0) = O(\log s)^{\mathtt{dth}-1}$ s.t.

$$\sum_{S \subseteq [n+d], |S| \le t} \hat{f}_S^2 > 1 - \epsilon_0 = 1/2.$$

Fix an $S = S_1 \cup \{i + n \mid i \in S_2\}$, with $|S| \le t$, where $S_1 \subseteq [n], S_2 \subseteq \{1, 2, \ldots, d\}$. We know that

$$\hat{f}_S = \langle f, \chi_S \rangle = 1 - 2 \Pr_u[f(u) \neq \chi_S(u)]$$

where u is uniformly drawn from $\{-1,1\}^{n+d}$.

For $a \in \{-1,1\}^{|S_1|}$, let X_a be the uniform distribution over $\{-1,1\}^n$ conditioned on $X_{S_1} = a$. For $b \in \{-1,1\}^{|S_2|}$, let R_b be the uniform distribution over $\{-1,1\}^d$ conditioned on $R_{S_2} = b$. So $\chi_S(x \circ r) = \prod_{i \in [|S_1|]} a_i \prod_{j \in [|S_2|]} b_j$ for $x \in \mathsf{supp}(X_a), r \in \mathsf{supp}(R_b)$. For special situations, saying $S_1 = \emptyset$ (or $S_2 = \emptyset$), let X_a (or R_b) be uniform.

As Ext is a strong (k,ϵ)-extractor, R_b only blows up the error by at most $2^{|S|}$.

Also note that X_a has entropy $n - |S_1| \geq n - t = n - O(\log s)^{\mathtt{dth}-1} \geq k = n - (c \log s)^{\mathtt{dth}-1}$ by choosing c large enough. So

$$\mathsf{dist}(f(X_a \circ R_b), U) \leq 2^{|S|}\epsilon,$$

where U is uniform over $\{-1, 1\}$.

So

$$\forall a \in \{-1,1\}^{|S_1|}, \forall b \in \{-1,1\}^{|S_2|}, |\Pr[f(X_a \circ R_b) \neq \prod_{i \in [|S_1|]} a_i \prod_{j \in [|S_2|]} b_j] - 1/2| \leq 2^{|S|}\epsilon.$$

Thus

$$\begin{aligned}
&|\Pr_u[f(u) \neq \chi_S(u)] - 1/2| \\
=&|\sum_{a \in \{-1,1\}^{|S_1|}} \sum_{b \in \{-1,1\}^{|S_2|}} \frac{1}{2^{|S|}} (\Pr[f(X_a \circ R_b) \neq \prod_{i \in [|S_1|]} a_i \prod_{j \in [|S_2|]} b_j] - 1/2)| \\
\leq &\sum_{a \in \{-1,1\}^{|S_1|}} \sum_{b \in \{-1,1\}^{|S_2|}} \frac{1}{2^{|S|}} |\Pr[f(X_a \circ R_b) \neq \prod_{i \in [|S_1|]} a_i \prod_{j \in [|S_2|]} b_j] - 1/2| \\
\leq &2^{|S|}\epsilon.
\end{aligned} \tag{1.5}$$

Hence

$$|\hat{f}_S| = |1 - 2\Pr_u[f(u) \neq \chi_S(u)]| = 2|\Pr_u[f(u) \neq \chi_S(u)] - 1/2| \leq 2^{|S|+1}\epsilon.$$

As a result,

$$1/2 \leq \sum_{S \subseteq [n+d], |S| \leq t} \hat{f}_S^2 \leq \sum_{i=0}^{t} \binom{n+d}{i} (2^{|S|+1}\epsilon)^2 \leq \sum_{i=0}^{t} \binom{n+d}{i} (2^{t+1}\epsilon)^2.$$

So

$$\epsilon \geq 2^{-(t+1)} \sqrt{\frac{1}{2\sum_{i=0}^{t} \binom{n+d}{i}}}.$$

As $\sum_{i=0}^{t} \binom{n+d}{i} \leq (\frac{e(n+d)}{t})^t = 2^{O(t\log(n+d))} = 2^{O(\log s)^{\mathtt{dth}-1}\log(n+d)}$, $\epsilon = 2^{-O(\log s)^{\mathtt{dth}-1}\log(n+d)}$.

□

1.4 The Basic Construction of Extractors in $\mathcal{AC}^0$

Our basic construction is based on the general idea of I-W generator [IW97]. In [Tre01], Trevisan showed that I-W generator is an extractor if we regard the string x drawn from the input (n,k)-source X as the truth table of a function f_x s.t. $f_x(\langle i \rangle), i \in [n]$ outputs the ith bit of x.

The construction of I-W generator involves a process of hardness amplifications from a worst-case hard function to an average-case hard function. There are mainly 3 amplification steps. Viola [Vio05b] summarizes these results in details, and we review them again. The first step is established by Babai et al. [Bab+93], which is an amplification from worst-case hardness to mildly average-case hardness.

Lemma 1.4.1 ([Bab+93])**.** *If there is a boolean function $f : \{0,1\}^l \to \{0,1\}$ which is 0-hard for circuit size $g = 2^{\Omega(l)}$ then there is a boolean function $f' : \{0,1\}^{\Theta(l)} \to \{0,1\}$ that is $1/\mathsf{poly}(l)$-hard for circuit size $g' = 2^{\Omega(l)}$.*

The second step is an amplification from mildly average-case hardness to constant average-case hardness, established by Impagliazzo [Imp95].

Lemma 1.4.2 ([Imp95])**.** *1. If there is a boolean function $f : \{0,1\}^l \to \{0,1\}$*

that is δ-hard for circuit size g where $\delta < 1/(16l)$, then there is a boolean function $f' : \{0,1\}^{3l} \to \{0,1\}$ that is $0.05\delta l$-hard for circuit size $g' = \delta^{O(1)} l^{-O(1)} g$.

$$f'(s,r) = \langle s, f(a_1) \circ f(a_2) \circ \cdots \circ f(a_l) \rangle$$

Here $|s| = l$, $|r| = 2l$ and $|a_i| = l, \forall i \in [l]$. Regarding r as a uniform random string, $a_1, \ldots, a_l$ are generated as pairwise independent random strings from the seed r.

2. *If there is a boolean function $f : \{0,1\}^l \to \{0,1\}$ that is δ-hard for circuit size g where $\delta < 1$ is a constant, then there is a boolean function $f' : \{0,1\}^{3l} \to \{0,1\}$ that is $1/2 - O(l^{-2/3})$-hard for circuit size $g' = l^{-O(1)} g$, where*

$$f'(s,r) = \langle s, f(a_1) \circ f(a_2) \circ \cdots \circ f(a_l) \rangle.$$

Here $|s| = l$, $|r| = 2l$ and $|a_i| = l, \forall i \in [l]$. Regarding r as a uniform random string, $a_1, \ldots, a_l$ are generated as pairwise independent random strings from the seed r.

The first part of this lemma can be applied for a constant number of times to get a function having constant average-case hardness. After that the second part is usually applied for only once to get a function with constant average-case hardness such that the constant is large enough (at least $1/3$).

The third step is an amplification from constant average-case hardness to even stronger average-case hardness, developed by Impagliazzo and Widgerson [IW97]. Their construction uses the following Nisan-Widgerson Generator [NW94] which is widely used in hardness amplification.

Definition 1.4.3 ((n,m,k,l)-design and Nisan-Widgerson Generator [NW94])**.** *A system of sets $S_1, S_2, \ldots, S_m \subseteq [n]$ is an (n,m,k,l)-design, if $\forall i \in [m], |S_i| = l$ and $\forall i, j \in [m], i \neq j, |S_i \cap S_j| \leq k$.*

Let $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ be an (n,m,k,l) design and $f : \{0,1\}^l \to \{0,1\}$ be a boolean function. The Nisan-Widgerson Generator is defined as $\mathsf{NW}_{f,\mathcal{S}}(u) = f(u|_{S_1}) \circ f(u|_{S_2}) \circ \cdots \circ f(u|_{S_m})$. Here $u|_{S_i} = u_{i_1} \circ u_{i_2} \circ \cdots \circ u_{i_m}$ assuming $S_i = \{i_1, \ldots, i_m\}$.

Nisan and Widgeson [NW94] showed that the (n,m,k,l)-design can be constructed efficiently.

Lemma 1.4.4 (Implicit in [NW94])**.** *For any $\alpha \in (0,1)$, for any large enough $l \in \mathbb{N}$, for any $m < \exp\{\frac{\alpha l}{4}\}$, there exists an $(n,m,\alpha l,l)$-design where $n = \lfloor \frac{10l}{\alpha} \rfloor$. This design can be computed in time polynomial of 2^n.*

As we need the parameters to be concrete (while in [NW94] they use big-O notations), we prove it again.

Proof. Our algorithm will construct these S_is one by one. For S_1, we can choose an arbitrary subset of $[n]$ of size αl.

First of all, S_1 can constructed by choosing l elements from $[n]$.

Assume we have constructed $S_1, \ldots, S_{i-1}$, now we construct S_i. We first prove that S_i exists. Consider a random subset of size l from $[n]$. Let $H_{i,j} = |S_i \cap S_j|$. We know that $\mathbf{E}H_{i,j} = l^2/n$. As $n = \lfloor \frac{10l}{\alpha} \rfloor \in [\frac{10l}{\alpha} - 1, \frac{10l}{\alpha}]$, $\mathbf{E}H_{i,j} \in [\frac{\alpha l}{10}, \frac{\alpha l}{10} + 1]$.

So $\Pr[H_{i,j} \geq \alpha l] \leq \Pr[H_{i,j} \geq (1+9)(\mathbf{E}H_{i,j} - 1)]$

By the Chernoff bound,

$$\begin{aligned}\Pr[H_{i,j} \geq 10(\mathbf{E}H_{i,j} - 1)] &\leq \Pr[H_{i,j} \geq 9\mathbf{E}H_{i,j}] \\ &\leq \exp\{-\frac{8\mathbf{E}H_{i,j}}{3}\} \\ &\leq \exp\{-\frac{4}{15}\alpha l\} \\ &\leq \exp\{-\frac{\alpha l}{4}\}\end{aligned} \tag{1.6}$$

By the union bound,

$$\Pr[\forall j = 1, \ldots, i-1, H_{i,j} \leq \alpha l] \geq 1 - m\exp\{-\frac{\alpha l}{4}\} > 0.$$

This proves that there exists a proper S_i. As there are n bits totally, we can find it in time polynomial of 2^n. □

The following is the third step of hardness amplification.

Lemma 1.4.5 (Implicit in [IW97])**.** *For any $\gamma \in (0, 1/30)$, if there is a boolean function $f : \{0,1\}^l \to \{0,1\}$ that is $1/3$-hard for circuit size $g = 2^{\gamma l}$, then there is a boolean function $f' : \{0,1\}^{l'=\Theta(l)} \to \{0,1\}$ that is $(1/2 - \epsilon)$-hard for circuit size $g' = \Theta(g^{1/4}\epsilon^2 l^{-2})$ where $\epsilon \geq (500l)^{1/3}g^{-1/12}$.*

$$f'(a, s, v_1, w) = \langle s, f(a|_{S_1} \oplus v_1) \circ f(a|_{S_2} \oplus v_2) \circ \cdots f(a|_{S_l} \oplus v_l)\rangle$$

Here $(S_1, \ldots, S_l)$ is an $(|a|, l, \gamma l/4, l)$-design where $|a| = \lfloor \frac{40l}{\gamma} \rfloor$. The vectors $v_1, \ldots, v_l$ are obtained by a random walk on an expander graph, starting at v_1 and walking according to w where $|v_1| = l, |w| = \Theta(l)$. The length of s is l. So $l' = |a| + |s| + |v_1| + |w| = \Theta(l)$.

The proof of Lemma 1.4.5 is in the Appendix.

The construction of the Impagliazzo Widgerson Generator [IW97] is as follows. Given the input $x \leftarrow X$, let $f : \{0,1\}^{\log n} \to \{0,1\}$ be such that $f(\langle a \rangle) = x_a, \forall a \in [n]$. Then we run the 3 amplification steps, Lemma 1.4.1, Lemma 1.4.2 (part1 for a constant number of times, part 2 for once) and Lemma 1.4.5 sequentially to get function f' from f. The generator $\mathsf{IW}(x,u) = \mathsf{NW}_{f',\mathcal{S}}(u)$. As pointed out by Trevisan [Tre01], the function IW is a (k,ϵ)-extractor. Let's call it the IW-Extractor. It is implicit in [Tre01] that the output length of the IW-Extractor is k^α and the statistical distance of $\mathsf{IW}(X,U)$ from uniform distributions is $\epsilon = 1/k^\beta$ for some $0 < \alpha, \beta < 1$. This can be verified by a detailed analysis of the IW-Extractor.

However, this construction is not in $\mathcal{AC}^0$ because the first amplification step is not in $\mathcal{AC}^0$.

Our basic construction is an adjustment of the IW-Extractor.

Construction 1.4.6. *For any $c_2 \in \mathbb{N}^+$ such that $c_2 \geq 2$ and any $k = \Theta(n/\log^{c_2-2} n)$, let X be an (n,k)-source . We construct a strong $(k, 2\epsilon)$ extractor $\mathsf{Ext}_0 : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ where $\epsilon = 1/n^\beta$, $\beta = 1/600$, $d = O(\log n)$, $m = k^{\Theta(1)}$. Let U be the uniform distribution of length d.*

1. *Draw x from X and u from U. Let $f_1 : \{0,1\}^{l_1} \to \{0,1\}$ be a boolean function such that $\forall i \in [2^{l_1}]$, $f_1(\langle i \rangle) = x_i$ where $l_1 = \log n$.*

2. *Run amplification step of Lemma 1.4.2 part 1 for c_2 times and run amplification step of Lemma 1.4.2 part 2 once to get function $f_2 : \{0,1\}^{l_2} \to \{0,1\}$ from f_1 where $l_2 = 3^{c_2+1} l_1 = \Theta(\log n)$.*

3. *Run amplification step Lemma 1.4.5 to get function $f_3 : \{0,1\}^{l_3} \to \{0,1\}$ from*

f_2 *where* $l_3 = \Theta(\log n)$.

4. *Construct function* Ext_0 *such that* $\mathsf{Ext}_0(x,u) = \mathsf{NW}_{f_3,\mathcal{S}}(u)$.

Here $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ *is a* $(d, m, \theta l_3, l_3)$*-design with* $\theta = l_1/(900 l_3)$, $d = \lfloor 10 l_3/\theta \rfloor$, $m = \lfloor 2^{\frac{\theta l_3}{4}} \rfloor = \lfloor n^{\frac{1}{3600}} \rfloor$.

Lemma 1.4.7. *In Construction 1.4.6,* Ext_0 *is a strong* $(k, 2\epsilon)$ *extractor.*

The proof follows from the "Bad Set" argument given by Trevisan [Tre01]. In Trevisan [Tre01] the argument is not explicit for strong extractors. Here our argument is explicit for proving that our construction gives a strong extractor.

Proof. We will prove that for every (n,k)-source X and for every $A : \{0,1\}^{d+m} \to \{0,1\}$ the following holds.

$$|\Pr[A(U_s \circ \mathsf{Ext}_0(X, U_s)) = 1] - \Pr[A(U) = 1]| \leq 2\epsilon$$

Here U_s is the uniform distribution over $\{0,1\}^d$ and U is the uniform distribution over $\{0,1\}^{d+m}$.

For every flat (n,k)-source X, and for every (fixed) function A, let's focus on a set $B \subseteq \{0,1\}^n$ such that $\forall x \in \mathsf{supp}(X)$, if $x \in B$, then

$$|\Pr[A(U_s \circ \mathsf{Ext}_0(x, U_s)) = 1] - \Pr[A(U) = 1]| > \epsilon.$$

According to Nisan and Widgerson [NW94], we have the following lemma.

Lemma 1.4.8 (Implicit in [NW94] [Tre01])**.** *If there exists an* A*-gate such that*

$$|\Pr[A(U_s \circ \mathsf{Ext}_0(x, U_s)) = 1] - \Pr[A(U) = 1]| > \epsilon,$$

then there is a circuit C_3 of size $O(2^{\theta l_3} m)$, using A-gates, that can compute f_3 correctly for $1/2 + \epsilon/m$ fraction of inputs.

Here A-gate is a special gate that can compute the function A.

By Lemma 1.4.8, there is a circuit C_3 of size $O(m2^{\theta l_3}) = O(2^{\frac{5\theta l_3}{4}}) = O(n^{1/720})$, using A-gates, that can compute f_3 correctly for $1/2 + \epsilon/m \geq 1/2 + 1/n^{1/400}$ fraction of inputs.

By Lemma 1.4.5, there is a circuit C_2, with A-gates, of size at most $\Theta(n^{\frac{1}{30}})$ which can compute f_2 correctly for at least $2/3$ fraction of inputs.

According to Lemma 1.4.2 and our settings, there is a circuit C_1, with A-gates, of size $n^{\frac{1}{30}} \operatorname{poly}\log n$ which can compute f_1 correctly for at least $1 - 1/(c_1 \log^{c_2} n)$ fraction of inputs for some constant $c_1 > 0$.

Next we give an upper bound on the size of B. $\forall x \in B$, assume we have a circuit of size $S = n^{1/30}\mathsf{poly}(\log n)$, using A-gates, that can compute at least $1 - 1/(c_1 \log^{c_2} n)$ fraction of bits of x. The total number of circuits, with A-gates, of size S is at most $2^{\Theta(mS \log S)} = 2^{n^{1/15}\mathsf{poly}(\log n)}$, as A is fixed and has fan-in $m + d = O(m)$. Each one of them corresponds to at most $\sum_{i=0}^{n/(c_1 \log^{c_2} n)} \binom{n}{i} \leq (e \cdot c_1 \log^{c_2} n)^{n/(c_1 \log^{c_2} n)} = 2^{O(n/\log^{c_2-1} n)}$ number of x. So

$$|B| \leq 2^{n^{1/15}\mathsf{poly}(\log n)} 2^{O(n/\log^{c_2-1} n)} = 2^{O(n/(\log^{c_2-1} n)}.$$

As X is an (n,k)-source with $k = \Theta(n/\log^{c_2-2} n)$,

$$\Pr[X \in B] \leq |B| \cdot 2^{-k} \leq \epsilon.$$

Then we know,

$$\begin{aligned}
&|\Pr[A(U_s \circ \mathsf{Ext}(X, U_s)) = 1] - \Pr[A(U) = 1]| \\
=&\sum_{x \in B} \Pr[X = x] |\Pr[A(U_s \circ \mathsf{Ext}(x, U_s)) = 1] - \Pr[A(U) = 1]| \\
&+ \sum_{x \notin B} \Pr[X = x] |\Pr[A(U_s \circ \mathsf{Ext}(x, U_s)) = 1] - \Pr[A(U) = 1]| \\
\leq& 2\epsilon.
\end{aligned} \tag{1.7}$$

□

Lemma 1.4.9. *The seed length of construction 1.4.6 is* $O(\log n)$.

Proof. We know that $l_1 = \log n, l_2 = 3^{c_2+1} l_1 = \Theta(\log n), l_3 = \Theta(\log n)$. Also $\mathcal{S}$ is a $(\lceil 10 l_3 / c \rceil = \Theta(l_3), m, c l_3, l_3)$-design. So $d = \lfloor 10 l_3 / c \rfloor = \Theta(l_3) = \Theta(\log n)$.

□

Lemma 1.4.10. *The function* Ext_0 *in Construction 1.4.6 is in* $\mathcal{AC}^0$. *The circuit depth is* $c_2 + 5$. *The locality is* $\Theta(\log^{c_2+2} n) = \mathsf{poly}(\log n)$.

Proof. First we prove that the locality is $\Theta(\log^{c_2+2} n)$.

By the construction of f_1, we know $f_1(\langle i \rangle)$ is equal to the ith bit of x.

Fix the seed u. According to Lemma 1.4.2 part 1, if we apply the amplification once to get f' from f, then $f'(s, r)$ depends on $f(w_1), f(w_2), \cdots, f(w_l)$, as

$$f'(s, r) = \langle s, f(w_1) \circ f(w_2) \circ \cdots \circ f(w_l) \rangle.$$

Here $l = O(\log n)$ is equal to the input length of f.

The construction in Lemma 1.4.2 part 2 is the same as that of Lemma 1.4.2 part 1.

As a result, if apply Lemma 1.4.2 part 1 for c_2 times and Lemma 1.4.2 part 2 for 1 time to get f_2 from f_1, the output of f_2 depends on $\Theta(\log^{c_2+1} n)$ bits of the input x.

According to Lemma 1.4.5, the output of f_3 depends on $f_2(a|_{S_1} \oplus v_1), f_2(a|_{S_2} \oplus v_2), \cdots, f_2(a|_{S_l} \oplus v_{l_2})$, as

$$f_3(a, s, v_1, w) = \langle s, f_2(a|_{S_1} \oplus v_1) \circ f_2(a|_{S_2} \oplus v_2) \circ \cdots f_2(a|_{S_l} \oplus v_{l_2}) \rangle$$

So the output of f_3 depends on $O(\log^{c_2+2} n)$ bits of the x.

So the overall locality is $O(\log^{c_2+2} n) = \mathsf{poly} \log n$.

Next we prove that the construction is in $\mathcal{AC}^0$.

The input of Ext_0 has two parts, x and u. Combining all the hardness amplification steps and the NW generator, we can see that essentially u is used for two purposes: to select some $t = \Theta(\log^{c_2+2}(n))$ bits (denote it as x') from x (i.e., provide t indices $u'_1, \ldots, u'_t$ in $[n]$), and to provide a vector s' of length t, finally taking the inner product of x' and the vector s'. Here although for each amplification step we do an inner product operation, the overall procedure can be realized by doing only one inner product operation.

Since u has $O(\log n)$ bits, s' can be computed from u by using a circuit of depth 2, according to Lemma 1.2.11 part 1.

Next we show that selecting x' from x using the indices can be computed by CNF/DNFs, of polynomial size, with inputs being x and the indices. The indices, $u'_i, i \in [t]$, are decided by u. Let's assume $\forall i \in [t], u'_i = h_i(u)$ for some deterministic functions $h_i, i \in [t]$. As $|u| = O(\log n)$, the indices can be computed by CNF/DNFs of polynomial size. Also $\forall i \in [t], f(u'_i)$ can be represented by a CNF/DNF when u'_i

is given. This is because

$$f(u_i') = \bigvee_{j=0}^{|x|} (I_{u_i'=j} \wedge x_j) = \bigwedge_{j=0}^{|x|} (I_{u_i' \neq j} \vee x_j).$$

Here I_e is the indicator function such that $I_e = 1$ if e is true and $I_e = 0$ otherwise. We know that $I_{u_i'=j}$ can be represented by a boolean formula with only AND and NOT gates, checking whether $u_i' = j$ bit by bit. Similarly $I_{u_i' \neq j}$ can be represented by a boolean formula with only OR and NOT gates, taking the negation of $I_{u_i'=j}$. As a result, this step can be computed by a circuit of depth 2.

So the computation of obtaining x' can be realized by a circuit of depth 3 by merging the gates between adjacent depths.

Finally we can take the inner product of two vectors x' and s' of length $t = \Theta(\log^{c_2+2}(n))$. By Lemma 1.2.11 part 2, we know that this computation can be represented by a poly-size circuit of depth $c_2 + 3$.

The two parts of computation can be merged together to be a circuit of depth $c_2 + 5$, as we can merge the last depth of the circuit obtaining x' and the first depth of the circuit computing the inner product. The size of the circuit is polynomial in n as both obtaining x' and the inner product operation can be realized by poly-size circuits.

□

By Construction 1.4.6, Lemma 1.4.7, Lemma 1.4.9, Lemma 1.4.10, we have the following theorem.

Theorem 1.4.11. *For any $c \in \mathbb{N}$, any $k = \Theta(n/\log^c n)$, there exists an explicit strong (k, ϵ)-extractor* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *in $\mathcal{AC}^0$ of depth $c + 7$, where $\epsilon = n^{-1/600}$, $d = O(\log n)$, $m = \lfloor n^{\frac{1}{3600}} \rfloor$ and the locality is $\Theta(\log^{c+4} n) =$*

$\mathsf{poly}\log n$.

We call this extractor the Basic-$\mathcal{AC}^0$-Extractor.

1.5 Error Reduction

By Theorem 1.4.11, for any $k = \frac{n}{\mathsf{poly}(\log n)}$, we have a (k, ϵ)-extractor in $\mathcal{AC}^0$, with $\epsilon = 1/n^{\beta}$ where β is a constant. In this section, we do error reduction to give an explicit (k, ϵ)-extractor in $\mathcal{AC}^0$ such that ϵ can be quasi-polynomially small.

We use two major techniques. First is the sample-then-extract method.

1.5.1 Sample-Then-Extract

We first analyze the sampling method which is well studied by Zuckerman [Zuc97], Vadhan [Vad04], Goldreich et al. [GVW15] and Healy [Hea08].

Definition 1.5.1 ([Vad04]). *A (μ_1, μ_2, γ)-averaging sampler is a function* $\mathsf{Samp} : \{0,1\}^r \to [n]^t$ *such that* $\forall f : [n] \to [0,1]$, *if* $\mathbf{E}_{i \in [n]}[f(i)] \geq \mu_1$, *then*

$$\Pr_{I \leftarrow \mathsf{Samp}(U_r)}[\frac{1}{t}\sum_{i \in I} f(i) < \mu_2] \leq \gamma.$$

The t samples generated by the sampler must be distinct.

Vadhan [Vad04] gives the following lemma on how to use samplers on weak sources.

Lemma 1.5.2 (Sample a Source [Vad04]). *Let* $0 < 3\tau \leq \delta \leq 1$. *If* $\mathsf{Samp} : \{0,1\}^r \to [n]^t$ *is a (μ_1, μ_2, γ)-averaging sampler for $\mu_1 = (\delta - 2\tau)/\log(1/\tau)$ and $\mu_2 = (\delta - 3\tau)/\log(1/\tau)$, then for every $(n, \delta n)$-source X, we have* $\mathsf{SD}(U \circ X_{\mathsf{Samp}(U_r)}, U \circ$

$W) \leq \gamma + 2^{-\Omega(\tau n)}$. *Here* U *is the uniform distribution over* $\{0,1\}^r$. *For every* a *in* $\{0,1\}^r$, *the random variable* $W|_{U=a}$ *is a* $(t, (\delta - 3\tau)t)$*-source.*

We mainly use the following samplers given by Healy [Hea08].

Theorem 1.5.3 ([Hea08] Theorem 3)**.** *For any* $n \in \mathbb{N}$, *any* $\mu \in (0,1], \varepsilon > \mu$, *there exists an* $(\mu, \mu - \epsilon, \gamma)$*-averaging sampler* $\mathsf{Samp} : \{0,1\}^r \to [n]^t$ *with seed length* $r = \log n + O(\log(1/\gamma)/\varepsilon^2)$ *and* $t = O(\log(1/\gamma)/\varepsilon^2)$ *which can be computed by* $\mathcal{NC}^1$ *circuits of size* $\mathsf{poly}(n, 1/\varepsilon, \log(1/\gamma))$.

Remark 1.5.4. *If* $\gamma = \Theta(2^{-\log^c n}), \varepsilon = \Theta(1/\log^a n)$, *then the sampler can be computed by* $\mathcal{AC}^0$ *circuitsof depth* $a + c + 1$ *by Lemma 1.2.11.*

For sample complexity (parameter t*), by Lemma 8.3 of [Vad04], we can modify the sampler and get a new sampler with the number of samples to be at least* t *while having the same seed length.*

After sampling, we use leftover hash lemma to do extraction.

Lemma 1.5.5 (Leftover Hash Lemma [IZ89])**.** *Let* X *be an* $(n', k = \delta n')$*-source. For any* $\Delta > 0$, *let* H *be a universal family of hash functions mapping* n' *bits to* $m = k - 2\Delta$ *bits. The distribution* $U \circ \mathsf{Ext}(X, U)$ *is at distance at most* $1/2^\Delta$ *to uniform distribution where the function* $\mathsf{Ext} : \{0,1\}^{n'} \times \{0,1\}^d \to \{0,1\}^m$ *chooses the* U*'th hash function* h_U *in* H *and outputs* $h_U(X)$.

We use the following universal hash function family $H = \{h_u, u \in \{0,1\}^{n'}\}$. *For every* u, *the hash function* $h_u(x)$ *equals to the last* m *bits of* $u \cdot x$ *where* $u \cdot x$ *is computed in* $\mathbb{F}_{2^{n'}}$.

Specifically, for any constant $a \in \mathbb{N}^+$, *for any* $n' = \Theta(\log^a n)$ *then* Ext *can be computed by an* $\mathcal{AC}^0$ *circuit of depth* $a + 1$.

Proof. The proof in [IZ89] has already shown that the universal hash function is a strong extractor. We only need to show that the hash functions can be computed in $\mathcal{AC}^0$.

Given a seed u, we need to compute $u \cdot x$ which is a multiplication in $\mathbb{F}_{2^{n'}}$. We claim that this can be done in $\mathcal{AC}^0$. Note that since the multiplication is in $\mathbb{F}_{2^{n'}}$, it is also a bi-linear function when regarding the two inputs as two n'-bit strings. Thus, each output bit is essentially the inner product over some input bits.

This shows that each output bit of $p \cdot q$ is an inner product of two vectors of n' dimension. As $n' = \Theta(\log^a n)$, by Lemma 1.2.11, this can be done in $\mathcal{AC}^0$ of depth $a+1$ and size $\mathsf{poly}(n)$. All the output bits can be computed in parallel. So $u \cdot x$ can be computed in $\mathcal{AC}^0$ of depth $a+1$ and size $\mathsf{poly}(n)$.

□

Theorem 1.5.6. *For any constant $\delta \in (0,1], a \in \mathbb{N}^+$ and any $\epsilon = 1/2^{-\Theta(\log^a n)}$, there exists an explicit construction of a $(k = \delta n, \epsilon)$-extractor $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ in $\mathcal{AC}^0$ of depth $2a+1$, where $d = O(\log(n/\varepsilon))$, $m = \Theta(\log(n/\varepsilon))$ and the locality is $O(\log(n/\epsilon))$.*

Proof. We follow the sample-then-extract procedure.

Let $\mathsf{Samp} : \{0,1\}^{r_s} \to \{0,1\}^t$ be a (μ_1, μ_2, γ)-averaging sampler following from Theorem 1.5.3. Let τ be a small enough constant, $\mu_1 = (\delta - 2\tau)/\log(1/\tau), \mu_2 = (\delta - 3\tau)/\log(1/\tau)$, $\gamma = 0.8\epsilon$. As a result, μ_1 is a constant and $\mu_2 = \alpha\mu_1$ for some constant $\alpha \in (0,1)$. For an (n,k)-source X, by Lemma 1.5.2, we have $\mathsf{SD}(R \circ X_{\mathsf{Samp}(R)}, R \circ W) \leq \gamma + 2^{-\Omega(\tau n)}$. Here R is a uniform random variable. For every r in $\{0,1\}^{r_s}$, the random variable $W|_{R=r}$ is a $(t, (\delta - 3\tau)t)$-source.

By Lemma 1.5.3, $r_s = \log n + O(\log(1/\gamma))$ and we can set $t = \Theta(\log(n/\varepsilon))$ to be large enough.

Let $\epsilon_1 = 0.1\epsilon$. We pick $m = O(\log(n/\varepsilon))$ to be such that $(\delta - 3\tau)t \geq m + 2\log(1/\epsilon_1)$. Let $\mathsf{Ext}_1 : \{0,1\}^t \times \{0,1\}^{d_1} \to \{0,1\}^m$ be a $((\delta - 3\tau)t, \epsilon_1)$-extractor following from Lemma 1.5.5. As a result,

$$\mathsf{SD}(U \circ \mathsf{Ext}_1(W, U), U') \leq \epsilon_1,$$

where U, U' are uniform distributions.

As a result, the sample-then-extract procedure gives an extractor of error

$$\gamma + 2^{-\Omega(\tau n)} + \epsilon_1 \leq 0.8\epsilon + 2^{-\Omega(\tau n)} + 0.1\epsilon.$$

As τ is a constant, $2^{-\Omega(\tau n)} \leq 0.1\epsilon$.

Thus the error of the extractor is at most ϵ.

The seed length is $r_s + d_1 = O(\log(n/\varepsilon))$.

The locality is $t = O(\log(n/\epsilon))$ because when the seed is fixed, we select t bits from X by sampling.

The sampler Samp is in $\mathcal{AC}^0$ of depth $a + 1$. The extractor Ext_1 is in $\mathcal{AC}^0$ of depth $a + 1$. So Ext is in $\mathcal{AC}^0$ of depth $2a + 1$.

□

In this way, we have an $\mathcal{AC}^0$ extractor which can have quasi-polynomial small errors.

1.5.2 Previous Error Reduction Techniques

Another tool we will be relying on is the error reduction method for extractors, given by Raz et al. [RRV99]. They give an error reduction method for poly-time extractors and we will adapt it to the $\mathcal{AC}^0$ settings.

Lemma 1.5.7 (G_x Property [RRV99])**.** *Let* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *be a* (k,ϵ)*-extractor with* $\epsilon < 1/4$*. Let* X *be any* $(n, k+t)$*-source. For every* $x \in \{0,1\}^n$*, there exists a set* G_x *such that the following holds.*

- *For every* $x \in \{0,1\}^n$*,* $G_x \subset \{0,1\}^d$ *and* $|G_x|/2^d = 1 - 2\epsilon$*.*
- $\mathsf{Ext}(X, G_X)$ *is within distance at most* 2^{-t} *from an* $(m, m - O(1))$*-source. Here* $\mathsf{Ext}(X, G_X)$ *is obtained by first sampling* x *according to* X*, then choosing* r *uniformly from* G_x*, and outputting* $\mathsf{Ext}(x, r)$*. We also denote* $\mathsf{Ext}(X, G_X)$ *as* $\mathsf{Ext}(X, U)|_{U \in G_X}$*.*

Raz et al. [RRV99] showed the following result.

Lemma 1.5.8 ([RRV99])**.** *Let* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *be a* (k,ϵ)*-extractor. Consider* $\mathsf{Ext}' : \{0,1\}^n \times \{0,1\}^{2d} \to \{0,1\}^{2m}$ *which is constructed in the following way.*

$$\mathsf{Ext}'(x,u) = \mathsf{Ext}(x,u_1) \circ \mathsf{Ext}(x,u_2)$$

Here $u = u_1 \circ u_2$*.*

For any $t \leq n-k$*, let* X *be an* $(n, k+t)$*-source . Let* U *be the uniform distribution of length* $2d$*.*

With probability at least $1 - O(\epsilon^2)$*,* $\mathsf{Ext}'(X,U)$ *is* 2^{-t}*-close to having entropy* $m - O(1)$*.*

Remark 1.5.9. *Here we briefly explain the result in lemma 1.5.8. The distribution of* $Y = \mathsf{Ext}'(X, U_1 \circ U_2)$ *is the convex combination of* $Y|_{U_1 \in G_X, U_2 \in G_X}$, $Y|_{U_1 \notin G_X, U_2 \in G_X}$, $Y|_{U_1 \in G_X, U_2 \notin G_X}$ *and* $Y|_{U_1 \notin G_X, U_2 \notin G_X}$. *That is*

$$\begin{aligned} Y = & I_{U_1 \in G_X, U_2 \in G_X} Y|_{U_1 \in G_X, U_2 \in G_X} + I_{U_1 \notin G_X, U_2 \in G_X} Y|_{U_1 \notin G_X, U_2 \in G_X} \\ & + I_{U_1 \in G_X, U_2 \notin G_X} Y|_{U_1 \in G_X, U_2 \notin G_X} + I_{U_1 \notin G_X, U_2 \notin G_X} Y|_{U_1 \notin G_X, U_2 \notin G_X}. \end{aligned} \tag{1.8}$$

Also we know that $\Pr[I_{U_1 \notin G_X, U_2 \notin G_X} = 1] = O(\epsilon^2)$. *As a result, according to Lemma 1.5.7, this lemma follows.*

Informally speaking, this means that if view $Y = \mathsf{Ext}'(X, U) = Y_1 \circ Y_2$, then with high probability either Y_1 or Y_2 is 2^t-close to having entropy $m - O(1)$.

We adapt this lemma by doing the extraction for any $t \in \mathbb{N}^+$ times instead of 2 times. We have the following result.

Lemma 1.5.10. *Let* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *be a* (k, ϵ)*-extractor. For any* $t \in \mathbb{N}^+$, *consider* $\mathsf{Ext}' : \{0,1\}^n \times \{0,1\}^{td} \to \{0,1\}^{tm}$ *which is constructed in the following way.*

$$\mathsf{Ext}'(x, u) = \mathsf{Ext}(x, u_1) \circ \mathsf{Ext}(x, u_2) \circ \cdots \circ \mathsf{Ext}(x, u_t)$$

Here $u = u_1 \circ u_2 \circ \cdots \circ u_t$.

For any $a \leq n - k$, *let* X *be an* $(n, k + a)$*-source. Let* $U = \bigcirc_{i=1}^{t} U_i$ *be the uniform distribution such that* $\forall i \in [t], |U_i| = d$.

1. *For* $S \subseteq [t]$, *let* $I_{S,X}$ *be the indicator such that* $I_{S,X} = 1$ *if* $\forall i \in S, U_i \in G_X, \forall j \notin S, U_j \notin G_X$ *and* $I_{S,X} = 0$ *otherwise. Here* G_X *is defined according to 1.5.7. The distribution of* $\mathsf{Ext}'(X, U)$ *is a convex combination of the distributions*

of $\mathsf{Ext}'(X,U)|_{I_{S,X}=1}, S \subseteq [t]$. *That is*

$$U \circ \mathsf{Ext}'(X,U) = \sum_{S \subseteq [t]} I_{S,X} U \circ \mathsf{Ext}'(X,U)|_{I_{S,X}=1}$$

2. *For every* $S \subseteq [t_1], S \neq \emptyset$, *there exists an* $i^* \in [t_1]$ *such that* $\mathsf{Ext}(X,U_{i^*})|_{I_{S,X}=1}$ *is* 2^{-a}*-close to having entropy* $m - O(1)$.

Proof. The first assertion is proved as the follows. By the definition of G_x of Lemma 1.5.7, for each fixed $x \in \mathsf{supp}(X)$, $\sum_{S \subseteq [t]} I_{S,x} = 1$ as for each i, $U_i \in G_x$ either happens or not. Also $I_{S,X}$ is a convex combination of $I_{S,x}, \forall x \in \mathsf{supp}(X)$. So $\sum_{S \subseteq [t]} I_{S,X} = \sum_{S \subseteq [t]} \sum_{x \in \mathsf{supp}(X)} I_{S,x} I_{X=x} = 1$. As a result, the assertion follows.

The second assertion is proved as the follows. For every $S \subseteq [t_1], S \neq \emptyset$, by the definition of $I_{S,X}$, there exists an $i^* \in [t_1]$, $U_{i^*} \in G_X$. By Lemma 1.5.7, $\mathsf{Ext}(X,U_{i^*})|_{U_{i^*} \in G_X} = \mathsf{Ext}(X,U_{i^*})|_{I_{S,X}=1}$ is 2^{-a}-close to having entropy $m - O(1)$.

□

1.5.3 The Construction

Finally we give the construction for error reduction of super-polynomially small errors.

Construction 1.5.11 (Error Reduction for Super-Polynomially Small Error)**.** *For any constant* $a \in \mathbb{N}^+$, *any constant* $c \in \mathbb{N}$, *any* $k = \Theta(n/\log^c n)$ *and any* $\epsilon = 1/2^{\Theta(\log^a n)}$, *we construct a strong* (k,ϵ)*-extractor* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *with* $m = k^{\Omega(1)}$.

- *Let* $\mathsf{Ext}_0 : \{0,1\}^{n_0=n} \times \{0,1\}^{d_0} \to \{0,1\}^{m_0}$ *be a* (k_0,ϵ_0)*-extractor following from Theorem 1.4.11 with* $k_0 \leq k - \Delta_1, \Delta_1 = \log(n/\epsilon)$, $\epsilon_0 = k^{-\Theta(1)}$, $d_0 =$

$\Theta(\log n)$, $m_0 = k^{O(1)}$.

- *Let* $\mathsf{Ext}_1 : \{0,1\}^{n_1 = m_0/t_2} \times \{0,1\}^{d_1} \to \{0,1\}^{m_1}$ *be a* (k_1, ϵ_1)*-extractor following from Theorem 1.5.6 where* $k_1 = 0.9n_1$, $\epsilon_1 = \epsilon/n$, $d_1 = O(\log(n/\epsilon))$, $m_1 = \Theta(\log(n/\epsilon))$.

- *Let* t_1 *be such that* $(2\epsilon_0)^{t_1} \leq 0.1\epsilon$. *(We focus on the case that* $\epsilon < \epsilon_0$. *If* $\epsilon \geq \epsilon_0$, *we set* Ext *to be* Ext_0.*)*

- *Let* $t_2 = m_0^{1/3}$.

Let X *be the input* (n, k) *source. Our construction is as follows.*

1. *Let* $R_1, R_2, \ldots, R_{t_1}$ *be independent uniform distributions such that for every* $i \in [t_1]$ *the length of* R_i *is* d_0. *Get* $Y_1 = \mathsf{Ext}_0(X, R_1), \ldots, Y_{t_1} = \mathsf{Ext}_0(X, R_{t_1})$.

2. *Get* $Y = Y_1 \circ Y_2 \circ Y_3 \circ \cdots \circ Y_{t_1}$.

3. *For each* $i \in [t_1]$, *let* $Y_i = Y_{i,1} \circ Y_{i,2} \circ \cdots \circ Y_{i,t_2}$ *such that for every* $j \in [t_2]$, $Y_{i,j}$ *has length* $n_1 = m_0/t_2$. *Let* $S_1, S_2, \ldots, S_{t_1}$ *be independent uniform distributions, each having length* d_1. *Get* $Z_{i,j} = \mathsf{Ext}_1(Y_{i,j}, S_i), \forall i \in [t_1], j \in [t_2]$. *Let* $Z_i = Z_{i,1} \circ Z_{i,2} \circ \cdots Z_{i,t_2}$.

4. *Let* $R = \bigcirc_i R_i, S = \bigcirc_i S_i$. *We get* $\mathsf{Ext}(X, U) = Z = \bigoplus_i^{t_1} Z_i$ *where* $U = R \circ S$.

Lemma 1.5.12. *Construction 1.5.11 gives a strong* (k, ϵ)*-extractor.*

In order to prove this Lemma, we need the following facts.

Lemma 1.5.13 (Chain Rule of Min-Entropy [Vad12])**.** *Let (X, Y) be a jointly distributed random variable with entropy k. The length of X is l. For every $\epsilon > 0$, with probability at least $1 - \epsilon$ over $x \leftarrow X$, $Y|_{X=x}$ has entropy $k - l - \log(1/\epsilon)$.*

Also there exists another source (X, Y') such that $\forall x \in \{0,1\}^l$, $Y'|_{X=x}$ has entropy $k - l - \log(1/\epsilon)$ and $\mathsf{SD}((X, Y), (X, Y')) \leq \epsilon$.

Lemma 1.5.14. *Let $X = X_1 \circ \cdots \circ X_t$ be an $(n, n - \Delta)$-source where for each $i \in [t]$, $|X_i| = n_1 = \omega(\Delta)$.*

Let $k_1 = n_1 - \Delta - \log(1/\epsilon_0)$ where ϵ_0 can be as small as $1/2^{0.9n_1}$.

Let $\mathsf{Ext}_1 : \{0,1\}^{n_1} \times \{0,1\}^{d_1} \to \{0,1\}^{m_1}$ be a strong (k_1, ϵ_1)-extractor.

Let $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be constructed as the following,

$$\mathsf{Ext}(X, U_s) = \mathsf{Ext}_1(X_1, U_s) \circ \cdots \circ \mathsf{Ext}_1(X_t, U_s).$$

Then Ext is a strong $(n - \Delta, \epsilon)$-extractor where $\epsilon = \mathsf{SD}(U_s \circ \mathsf{Ext}(X, U_s), U) \leq t(\epsilon_0 + \epsilon_1)$.

Proof. We prove by induction over the block index i.

For simplicity, let $\tilde{X}_i = X_1 \circ \cdots \circ X_i$ for every i. We slightly abuse the notation Ext here so that $\mathsf{Ext}(\tilde{X}_i, U_s) = \mathsf{Ext}_1(X_1, U_s) \circ \cdots \circ \mathsf{Ext}_1(X_i, U_s)$ denotes the extraction for the first i blocks.

For the first block, we know $H_\infty(X_1) = n_1 - \Delta$. According to the definition of Ext_1,

$$\mathsf{SD}(U_s \circ \mathsf{Ext}_1(X_1, U_s), U) \leq \epsilon_1 \leq (\epsilon_0 + \epsilon_1).$$

Assume for the first $i - 1$ blocks, $\mathsf{SD}(U_s \circ \mathsf{Ext}_1(\tilde{X}_{i-1}, U_s), U) \leq (i - 1)(\epsilon_0 + \epsilon_1)$. Consider $\tilde{X}_i$. By Lemma 1.5.13, we know that there exists X_i' such that

$\mathsf{SD}(\tilde{X}_i, \tilde{X}_{i-1} \circ X'_i) \leq \epsilon_0$, where X'_i is such that $\forall \tilde{x}_{i-1} \in \mathsf{supp}(\tilde{X}_{i-1}), H_\infty(X'_i | \tilde{X}_{i-1} = \tilde{x}_{i-1}) \geq n_1 - \Delta - \log(1/\epsilon_0)$. So according to Lemma 1.2.4 part 2, as $U_s \circ \mathsf{Ext}(\tilde{X}_{i-1}, U_s) \circ \mathsf{Ext}_1(X_i, U_s)$ is a convex combination of $u \circ \mathsf{Ext}(\tilde{X}_{i-1}, u) \circ \mathsf{Ext}_1(X_i, u), \forall u \in \mathsf{supp}(U_s)$ and $U_s \circ \mathsf{Ext}(\tilde{X}_{i-1}, U_s) \circ \mathsf{Ext}_1(X'_i, U_s)$ is a convex combination of $u \circ \mathsf{Ext}(\tilde{X}_{i-1}, u) \circ \mathsf{Ext}_1(X'_i, u), \forall u \in \mathsf{supp}(U_s)$, we have

$$\mathsf{SD}(U_s \circ \mathsf{Ext}(\tilde{X}_{i-1}, U_s) \circ \mathsf{Ext}_1(X_i, U_s), U_s \circ \mathsf{Ext}(\tilde{X}_{i-1}, U_s) \circ \mathsf{Ext}_1(X'_i, U_s)) \leq \mathsf{SD}(\tilde{X}_i, \tilde{X}_{i-1} \circ X'_i) \leq \epsilon_0$$

According to the assumption, Lemma 1.2.9 and the triangle inequality of Lemma 1.2.4, we have the following.

$$\begin{aligned}
&\mathsf{SD}(U_s \circ \mathsf{Ext}(\tilde{X}_{i-1}, U_s) \circ \mathsf{Ext}_1(X_i, U_s), U) \\
\leq &\mathsf{SD}(U_s \circ \mathsf{Ext}(\tilde{X}_{i-1}, U_s) \circ \mathsf{Ext}_1(X_i, U_s), U_s \circ \mathsf{Ext}(\tilde{X}_{i-1}, U_s) \circ \mathsf{Ext}_1(X'_i, U_s)) \\
&+ \mathsf{SD}(U_s \circ \mathsf{Ext}(\tilde{X}_{i-1}, U_s) \circ \mathsf{Ext}_1(X'_i, U_s), U) \\
\leq &\epsilon_0 + (i-1)(\epsilon_0 + \epsilon_1) + \epsilon_1 \\
= &i(\epsilon_0 + \epsilon_1)
\end{aligned} \tag{1.9}$$

The first inequality is due to the triangle property of Lemma 1.2.4. For the second inequality, first we have already shown that $\mathsf{SD}(U_s \circ \mathsf{Ext}(\tilde{X}_{i-1}, U_s) \circ \mathsf{Ext}_1(X_i, U_s), U_s \circ \mathsf{Ext}(\tilde{X}_{i-1}, U_s) \circ \mathsf{Ext}_1(X'_i, U_s)) \leq \epsilon_0$. Second, as $\tilde{X}_{i-1} \circ X'_i$ is a $((i-1)n_1 - \Delta, n_1 - \Delta - \log(1/\epsilon_0))$-block source, by our assumption and Lemma 1.2.9, $\mathsf{SD}(U_s \circ \mathsf{Ext}(\tilde{X}_{i-1}, U_s) \circ \mathsf{Ext}_1(X'_i, U_s), U) \leq (i-1)(\epsilon_0 + \epsilon_1) + \epsilon_1$. This proves the induction step.

As a result, $\mathsf{SD}(U_s \circ \mathsf{Ext}(X, U_s), U) \leq (\epsilon_0 + \epsilon_1)t$. □

Next we prove Lemma 1.5.12.

Proof of Lemma 1.5.12. Let G_X be defined by Lemma 1.5.7 on X and Ext_0.

For any $T \subseteq [t_1]$, let $I_{T,X}$ be the indicator such that $I_{T,X} = 1$, if $\forall i \in T, R_i \in G_X, \forall i \notin T, R_i \notin G_X$ and $I_{T,X} = 0$, otherwise. By Lemma 1.5.10,

$$\begin{aligned} & R \circ Y \\ =& \sum_{T \subseteq [t_1]} I_{T,X}(R \circ Y|_{I_{T,X}=1}) \\ =& I_{\emptyset,X}(R \circ Y|_{I_{\emptyset,X}=1}) + (1 - I_{\emptyset,X})(R \circ Y|_{I_{\emptyset,X}=0}) \\ =& I_{\emptyset,X}(R \circ Y|_{I_{\emptyset,X}=1}) + \sum_{T \subseteq [t_1], T \neq \emptyset} I_{T,X}(R \circ Y|_{I_{T,X}=1}) \end{aligned} \tag{1.10}$$

Fixing a set $T \subseteq [t_1], T \neq \emptyset$, by Lemma 1.5.10, there exists an $i^* \in [t_1]$ such that $R_{i^*} \in G_X$ and $R \circ Y|_{I_{T,X}=1}$ is $2^{-\Delta_1}$-close to

$$R' \circ A \circ W \circ B = \bigcirc_i R'_i \circ A \circ W \circ B$$

Here $W = Y_{i^*}|_{R_i \in G_X}$ has entropy at least $m_0 - O(1)$. Also, A, B and $R'_i, i = 1, 2, \ldots, t$ are some random variables where $A = (i^* - 1)m_1, |B| = (t - i^*)m_1$ and $\forall i \in [t], |R'_i| = d_1$. In fact, $R' = R|_{I_{T,X}=1}$.

According to our construction, next step we view $A \circ W \circ B$ as having $t_1 t_2$ blocks of block size n_1. We apply the extractor Ext_1 on each block. Although for all blocks the extractions are conducted simultaneously, we can still view the procedure as first extracting A and B, then extracting W. Assume for A, after extraction by using

seed S_A, it outputs A'. Also for B, after extraction by using seed S_B, it outputs B'. So after extracting A and B, we get $A' \circ W \circ B'$. The length of $A' \circ B'$ is at most $t_1 m_0 m_1 / n_1 = t_1 t_2 m_1 = \Theta(t_1 t_2 \log n)$, as $n_1 = m_0 / t_2$.

We know that $n_1 = m_0/t_2 = m_0^{2/3} \geq 10 t_1 t_2 m_1 = m_0^{1/3} \mathsf{poly}(\log n)$. Also according to Lemma 1.5.13, $R' \circ S_A \circ S_B \circ A' \circ W \circ B'$ is ϵ'-close to $R' \circ S_A \circ S_B \circ A' \circ W' \circ B'$ such that for every $r' \in \mathsf{supp}(R'), a \in \mathsf{supp}(A'), b \in \mathsf{supp}(B'), s_A \in \mathsf{supp}(S_A), s_B \in \mathsf{supp}(S_B)$, conditioned on $R' = r', S_A = s_A, S_B = s_B, A' = a, B' = b$, W' has entropy at least $n_1 - O(\log n) - t_1 t_2 m_1 - \log(1/\epsilon') = n_1 - \Delta_2$ where $\Delta_2 = O(\log n) + t_1 t_2 m_1 + \log(1/\epsilon') = O(m_0^{1/3} \log n)$. Here ϵ' can be as small as $2^{-k^{\Omega(1)}}$. That is

$$\forall r' \in \mathsf{supp}(R'), a \in \mathsf{supp}(A'), b \in \mathsf{supp}(B'), s_A \in \mathsf{supp}(S_A), s_B \in \mathsf{supp}(S_B),$$

$$H_\infty(W'|_{R'=r', S_A=s_A, S_B=s_B, A'=a, B'=b}) \geq n_1 - \Delta_2. \tag{1.11}$$

Let $\mathsf{Ext}'_1(W', S_{i^*}) = \bigcirc_{i \in [t_2]} \mathsf{Ext}_1(W'_i, S_{i^*})$ where $W' = \bigcirc_{i \in [t_2]} W'_i$ and $\forall i \in [t_2], |W'_i| = n_1$. By Lemma 1.5.14, as $k_1 = 0.9 n_1 \leq n_1 - \Delta_2$, $S_{i^*} \circ \mathsf{Ext}'_1(W', S_{i^*})|_{R'=r', S_A=s_A, S_B=s_B, A'=}$ is $(\epsilon'_0 + \epsilon_1) t_2$-close to uniform distributions where ϵ'_0 can be as small as $2^{-k^{\Omega(1)}}$.

As a result, we have the following.

$$
\begin{aligned}
&\mathsf{SD}(U \circ \mathsf{Ext}(X,U), U') \\
=&\mathsf{SD}(R \circ S \circ \mathsf{Ext}(X,U), R \circ S \circ \tilde{U}) \\
=&\mathsf{SD}(I_{\emptyset,X}(R \circ S \circ \mathsf{Ext}(X,U)|_{I_{\emptyset,X}=1}), I_{\emptyset,X}(R \circ S \circ \tilde{U}|_{I_{\emptyset,X}=1})) \\
&+ \mathsf{SD}((1-I_{\emptyset,X})(R \circ S \circ \mathsf{Ext}(X,U)|_{I_{\emptyset,X}=0}), (1-I_{\emptyset,X})(R \circ S \circ \tilde{U}|_{I_{\emptyset,X}=0})) \\
=&\Pr[I_{\emptyset,X}=1]\mathsf{SD}(R \circ S \circ \mathsf{Ext}(X,U)|_{I_{\emptyset,X}=1}, R \circ S \circ \tilde{U}|_{I_{\emptyset,X}=1}) \\
=&(2\epsilon_0)^{t_1}\mathsf{SD}(R \circ S \circ \mathsf{Ext}(X,U)|_{I_{\emptyset,X}=1}, R \circ S \circ \tilde{U}|_{I_{\emptyset,X}=1}) \\
&+ \mathsf{SD}((1-I_{\emptyset,X})(R \circ S \circ \mathsf{Ext}(X,U)|_{I_{\emptyset,X}=0}), (1-I_{\emptyset,X})(R \circ S \circ \tilde{U}|_{I_{\emptyset,X}=0}))
\end{aligned}
\tag{1.12}
$$

As

$$(2\epsilon_0)^{t_1}\mathsf{SD}(R \circ S \circ \mathsf{Ext}(X,U)|_{I_{\emptyset,X}=1}, R \circ S \circ \tilde{U}|_{I_{\emptyset,X}=1}) \leq (2\epsilon_0)^{t_1}$$

let's focus on $\mathsf{SD}((1-I_{\emptyset,X})(R\circ S\circ \mathsf{Ext}(X,U)|_{I_{\emptyset,X}=0}),(1-I_{\emptyset,X})(R\circ S\circ \tilde{U}|_{I_{\emptyset,X}=0}))$.

$$
\begin{aligned}
&\mathsf{SD}((1-I_{\emptyset,X})(R\circ S\circ \mathsf{Ext}(X,U)|_{I_{\emptyset,X}=0}),(1-I_{\emptyset,X})(R\circ S\circ \tilde{U}|_{I_{\emptyset,X}=0}))\\
=&\mathsf{SD}(\sum_{T\subseteq[t_1],T\neq\emptyset} I_{T,X}(R\circ S\circ \mathsf{Ext}(X,U)|_{I_{T,X}=1}),\sum_{T\subseteq[t_1],T\neq\emptyset} I_{T,X}(R\circ S\circ \tilde{U}|_{I_{T,X}=1}))\\
=&\sum_{T\subseteq[t_1],T\neq\emptyset}\Pr[I_{T,X}=1]\mathsf{SD}(R\circ S\circ \mathsf{Ext}(X,U)|_{I_{T,X}=1},R\circ S\circ \tilde{U}|_{I_{T,X}=1})\\
\leq&\sum_{T\subseteq[t_1],T\neq\emptyset}\Pr[I_{T,X}=1](2^{-\Delta_1}+\mathsf{SD}(R'\circ S\circ(A'\oplus \mathsf{Ext}'_1(W,S_{i^*})\oplus B'),R'\circ S\circ \tilde{U}))\\
=&(1-(2\epsilon_0)^{t_1})(2^{-\Delta_1}+\mathsf{SD}(R'\circ S\circ(A'\oplus \mathsf{Ext}'_1(W,S_{i^*})\oplus B'),R'\circ S\circ \tilde{U}))\\
\leq&2^{-\Delta_1}+\mathsf{SD}(R'\circ S\circ(A'\oplus \mathsf{Ext}'_1(W,S_{i^*})\oplus B'),R'\circ S\circ \tilde{U})\\
\leq&2^{-\Delta_1}+\mathsf{SD}(R'\circ S\circ(A'\oplus \mathsf{Ext}'_1(W,S_{i^*})\oplus B'),R'\circ S\circ(A'\oplus \mathsf{Ext}'_1(W',S_{i^*})\oplus B'))\\
&+\mathsf{SD}(R'\circ S\circ(A'\oplus \mathsf{Ext}'_1(W',S_{i^*})\oplus B'),R'\circ S\circ \tilde{U})\\
\leq&2^{-\Delta_1}+\epsilon'+\mathsf{SD}(R'\circ S\circ(A'\oplus \mathsf{Ext}'_1(W',S_{i^*})\oplus B'),R'\circ S\circ \tilde{U})\\
\leq&2^{-\Delta_1}+\epsilon'+(\epsilon'_0+\epsilon_1)t_2
\end{aligned}
\tag{1.13}
$$

Here $U,U',\tilde{U}$ are uniform distributions. In the second equation, $I_{\emptyset,X}$ is the indicator such that $I_{\emptyset,X}=1$ if $\forall i\in[t_1],R_i\notin G_X$ where G_X is defined by Lemma 1.5.7 on X and Ext_0. For the first inequality, we need to show that

$$\mathsf{SD}(R\circ S\circ \mathsf{Ext}(X,U)|_{I_{T,X}=1},R'\circ S\circ(A'\oplus \mathsf{Ext}'_1(W,S_{i^*})\oplus B'))\leq 2^{-\Delta_1}.$$

We know that for every $s \in \mathsf{supp}(S)$, by Lemma 1.2.4 part 2,

$$\begin{aligned}&\mathsf{SD}(R \circ S \circ \mathsf{Ext}(X,U)|_{I_{T,X}=1,S=s}, R' \circ S \circ (A' \oplus \mathsf{Ext}'_1(W, S_{i^*}) \oplus B')|_{S=s})\\ \leq &\mathsf{SD}(R \circ Y|_{I_{T,X}=1}, R' \circ A \circ W \circ B)\\ \leq &2^{-\Delta_1}.\end{aligned} \tag{1.14}$$

Here $R \circ S \circ \mathsf{Ext}(X,U)|_{I_{T,X}=1,S=s} = h(R \circ Y|_{I_{T,X}=1})$ for some deterministic function h as $S = s$ is fixed. Also $R' \circ S \circ (A' \oplus \mathsf{Ext}'_1(W, S_{i^*}) \oplus B')|_{S=s} = h(R' \circ A \circ W \circ B)$ for the same reason. As a result,

$$\begin{aligned}&\mathsf{SD}(R \circ S \circ \mathsf{Ext}(X,U)|_{I_{T,X}=1}, R' \circ S \circ (A' \oplus \mathsf{Ext}'_1(W, S_{i^*}) \oplus B'))\\ = &\sum_{s \in \mathsf{supp}(S)} \Pr[S=s]\mathsf{SD}(R \circ S \circ \mathsf{Ext}(X,U)|_{I_{T,X}=1,S=s}, R' \circ S \circ (A' \oplus \mathsf{Ext}'_1(W, S_{i^*}) \oplus B')|_{S=s})\\ \leq &2^{-\Delta_1}.\end{aligned} \tag{1.15}$$

The third inequality holds by the triangle property of Lemma 1.2.4 part 1. The 4th inequality holds because by Lemma 1.2.4 part 2,

$$\begin{aligned}&\mathsf{SD}(R' \circ S \circ (A' \oplus \mathsf{Ext}'_1(W, S_{i^*}) \oplus B')|_{S=s}, R' \circ S \circ (A' \oplus \mathsf{Ext}'_1(W', S_{i^*}) \oplus B')|_{S=s})\\ \leq &\mathsf{SD}(R' \circ S \circ A \circ W \circ B, R' \circ S \circ A \circ W' \circ B)\\ \leq &\epsilon'.\end{aligned} \tag{1.16}$$

.

As a result, the total error is at most

$$(2\epsilon_0)^{t_1} + (2^{-\Delta_1} + \epsilon' + (\epsilon'_0 + \epsilon_1)t_2)$$

We can set $\epsilon' = 0.1\epsilon, \epsilon'_0 = \epsilon/n$ so that $(2^{-\Delta_1} + \epsilon' + (\epsilon'_0 + \epsilon_1)t_2) \leq 0.1\epsilon$. As $(2\epsilon_0)^{t_1} < 0.1\epsilon$, we know $\mathsf{SD}(U \circ \mathsf{Ext}(X,U), U') \leq \epsilon$.

□

Lemma 1.5.15. *In Construction 1.5.11, the output length of* Ext *is* $m = \Omega(n^{10800}\log(n/\epsilon))$.

Proof. The output length is equal to $t_2 \times m_1 = m_0^{1/3}\Omega(\log(n/\epsilon)) = \Omega(n^{1/10800}\log(n/\epsilon))$

□

Lemma 1.5.16. *In Construction 1.5.11, the function* Ext *can be realized by a circuit of depth* $3a + c + 7$. *Its locality is* $O(\log^{2a+c+4} n)$.

Proof. By Theorem 1.4.11 and Lemma 1.5.6, both Ext_0 and Ext_1 in our construction are in $\mathcal{AC}^0$. For Ext_0, it can be realized by circuits of depth $c + 7$. For Ext_1, it can be realized by circuits of depth $2a + 1$.

In the first and second steps of Construction 1.5.11, we only run Ext_0 for t_1 times in parallel. So the computation can be realized by circuits of depth $c + 7$

For the third step, we run Ext_1 for $t_1 t_2$ times in parallel, which can be realized by circuits of depth $2a + 1$.

The last step, according to Lemma 1.2.11, taking the XOR of $O(\log(1/\epsilon))$ bits can be realized by circuits of depth $a + 1$. Each bit of Z is the XOR of t_1 bits and all the bits of Z can be computed in parallel. So the computations in this step can be realized by circuits of depth $a + 1$.

Now we merge the three parts of circuits together. As the circuits between each parts can be merged by deleting one depth, our construction can be realized by circuits of depth

$$(c + 7) + 2a + 1 + a + 1 - 2 = 3a + c + 7.$$

For the locality, by Theorem 1.4.11, the locality of Ext_0 is $O(\log^{c+4} n)$. By Lemma 1.5.6, the locality of Ext_1 is $O(\log(n/\epsilon))$. So each bit of Z is related with at most $t_1 \times O(\log(n/\epsilon)) \times O(\log^{c+4} n) = O(\log^{2a+c+4} n)$ bits of X.

□

Lemma 1.5.17. *In Construction 1.5.11, $d = O(\log n + \frac{\log(n/\epsilon)\log(1/\epsilon)}{\log n})$.*

Proof. In Construction 1.5.11, as

$$U = R \circ S = \bigcirc_i R_i \circ \bigcirc_i S_i,$$

$|U| = O(t_1 d_0 + t_1 d_1)$. By the definitions of Ext_0 and Ext_1, we know that $d_0 = O(\log n)$ and $d_1 = O(\log(n/\epsilon))$. Also we know that $t_1 = O(\log(1/\epsilon)/\log n)$ because $\epsilon_0 = n^{-\Theta(1)}$, $(2\epsilon_0)^{t_1} \leq 0.1\epsilon$. Note that we have to compute Ext_0 for at least once. So $d = O(\log n + \frac{\log(n/\epsilon)\log(1/\epsilon)}{\log n})$. □

Theorem 1.5.18. *For any constant $a, c \in \mathbb{N}$, any $k = \Theta(n/\log^c n)$ and any $\epsilon = 1/2^{\Theta(\log^a n)}$, there exists an explicit construction of a strong (k,ϵ)-extractor $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ in $\mathcal{AC}^0$ of depth $3a + c + 7$, where $d = O(\log n + \frac{\log(n/\epsilon)\log(1/\epsilon)}{\log n})$, $m = \Omega(n^{1/10800}\log(n/\epsilon)) = k^{\Omega(1)}$ and the locality is $(\log n)^{2a+c+4}$.*

Proof. It follows from Construction 1.5.11, Lemma 1.5.12, Lemma 1.5.15 , Lemma 1.5.16 and Lemma 1.5.17. □

If we do not require the extractor to be in $\mathcal{AC}^0$, we can get an extractor with low locality while having very small error. The construction is similar to Construction 1.5.11. The proof is also in the same way as that of Theorem 1.5.18. So we directly give the result.

Theorem 1.5.19. *For any constant $c \in \mathbb{N}$, any $k = \Theta(\frac{n}{\log^c n})$, there exists an explicit construction of a strong (k, ϵ)-extractor $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, where ϵ can be as small as $2^{-k^{\Omega(1)}}$, $d = \Theta(\log n + \frac{\log(n/\epsilon)\log(1/\epsilon)}{\log n})$, $m = \Omega(n^{\frac{1}{10800}} \log(\frac{n}{\epsilon}))$ $= k^{\Omega(1)}$ and the locality is $\log^2(1/\epsilon)\log^{c+O(1)} n$.*

1.6 Output Length Stretching

In this section, we show how to extract $(1-\gamma)k$ bits for any constant $\gamma > 0$.

1.6.1 Pre-sampling

By Theorem 1.5.18, we have a (k, ϵ)-extractor in $\mathcal{AC}^0$ for any $k = n/\mathsf{poly}(\log n)$ and any $\epsilon = 1/\mathsf{poly}(n)$. We use pre-sampling to increase the output length.

Zuckerman [Zuc97] gives sampler (oblivious sampler) constructions from extractors.

Definition 1.6.1 ([Zuc97]). *An $(n, m, t, \gamma, \epsilon)$-oblivious sampler is a deterministic function $\mathsf{Samp} : \{0,1\}^n \to (\{0,1\}^m)^t$ such that $\forall f : \{0,1\}^m \to [0,1]$,*

$$\Pr_{I \leftarrow \mathsf{Samp}(U_r)}[|\frac{1}{t}\sum_{i \in I} f(i) - \mathbf{E}f| > \epsilon] \leq \gamma.$$

The following lemma explicitly gives a construction of oblivious samplers using extractors.

Lemma 1.6.2 ([Zuc97]). *If there is an explicit $(k = \delta n, \epsilon)$-extractor $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, then there is an explicit $(n, m, t = 2^d, \gamma = 2^{1-(1-\delta)n}, \epsilon)$-oblivious sampler.*

The sampler is constructed as follows. Given a seed x of length n, the $t = 2^d$

samples are $\mathsf{Ext}(x,u)$, $\forall u \in \{0,1\}^d$.

As a result, we can construct the following samplers.

Lemma 1.6.3. *For any* $a \in \mathbb{N}^+$ *, let* γ *be any* $1/2^{\Theta(\log^a n)}$.

- *For any* $c \in \mathbb{N}$, *let* ϵ *be any* $\Theta(1/\log^c n)$. *There exists an explicit* $(O(\log(1/\gamma)), \log n, t, \gamma, \epsilon)$*-oblivious sampler for any integer* $t \in [t_0, n]$ *with* $t_0 = \mathsf{poly}(\log n)$.
- *For any constant* α *in* $(0,1)$, *any* $c \in \mathbb{N}$, *any* $\mu = \Theta(1/\log^c n)$, *there exists an explicit* $(\mu, \alpha\mu, \gamma)$*-averaging sampler* $\mathsf{Samp} : \{0,1\}^{\Theta(\log^a n)} \to [n]^t$ *in* $\mathcal{AC}^0$ *of circuit depth* $a+2$, *for any integer* $t \in [t_0, n]$ *with* $t_0 = \mathsf{poly}(\log n)$.

  Specifically, if $c = 0$, t *can be any integer in* $[t_0, n]$ *with* $t_0 = (\log n)^{\Theta(a)}$.

Proof. Let $k = \log^a n$. For any $\epsilon = \Theta(1/\log^c n)$, let's consider a (k,ϵ)-extractor $\mathsf{Ext} : \{0,1\}^{n'=c_0 \log^a n} \times \{0,1\}^d \to \{0,1\}^{\log n}$ for some constant c_0, following Lemma 1.2.8. Here we make one modification. We replace the last d bits of the output with the seed. We can see in this way, Ext is still an extractor.

Here the entropy rate is $\delta = 1/c_0$ which is a constant. According to Lemma 1.2.8, we know that, d can be $\Theta(\frac{\log^2(n'/\epsilon)}{\log n'}) = \Theta(\log\log n)$.

For the first assertion, according to Lemma 1.6.2, there exists an explicit construction of a $(c\log^a n, \log n, t, \gamma, \epsilon)$-oblivious sampler where $\gamma = 2^{1-(1-2/c)(c\log^a n)}$. As we can increase the seed length to $\log n$ by padding uniform random bits, t can be any integer in $[t_0, n]$ with $t_0 = 2^{\Theta(\frac{\log^2(n'/\epsilon)}{\log n'})} = \mathsf{poly}(\log n)$. As c_0 can be any large enough constant, γ can be $1/2^{\Theta(\log^a n)}$.

Next we prove the second assertion.

According to the definition of oblivious sampler, we know that $\forall f : [n] \to [0,1]$,

$$\Pr_{I \leftarrow \mathsf{Samp}(U)}[|\frac{1}{t}\sum_{i \in I} f(i) - \mathbf{E}f| > \epsilon] \leq \gamma.$$

Next we consider the definition of averaging sampler.

Let $(1-\alpha)\mu = \epsilon$. As $\mu = \Theta(1/\log^c n)$, $\epsilon = \Theta(1/\log^c n)$. For any $f : [n] \to [0,1]$ such that $\mu \leq \mathbf{E}f$, we have the following inequalities, where Samp is a $(c\log^a n, \log n, t, \gamma, \epsilon)$-oblivious sampler.

$$\begin{aligned}
&\Pr_{I \leftarrow \mathsf{Samp}(U)}[\frac{1}{t}\sum_{i \in I} f(i) < \alpha\mu] \\
=&\Pr_{I \leftarrow \mathsf{Samp}(U)}[\frac{1}{t}\sum_{i \in I} f(i) < \mu - \epsilon] \\
=&\Pr_{I \leftarrow \mathsf{Samp}(U)}[\mu - \frac{1}{t}\sum_{i \in I} f(i) > \epsilon] \\
\leq&\Pr_{I \leftarrow \mathsf{Samp}(U)}[\mathbf{E}f - \frac{1}{t}\sum_{i \in I} f(i) > \epsilon] \\
\leq&\Pr_{I \leftarrow \mathsf{Samp}(U)}[|\frac{1}{t}\sum_{i \in I} f(i) - \mathbf{E}f| > \epsilon] \\
\leq&\gamma
\end{aligned} \tag{1.17}$$

The first inequality holds because if the event that $\mu - \frac{1}{t}\sum_{i \in I} f(i) > \epsilon$ happens, then the event that $\mathbf{E}f - \frac{1}{t}\sum_{i \in I} f(i) > \epsilon$ will happen, as $\mu \leq \mathbf{E}f$. The second inequality is because $\mathbf{E}f - \frac{1}{t}\sum_{i \in I} f(i) \leq |\mathbf{E}f - \frac{1}{t}\sum_{i \in I} f(i)|$. So if $\mathbf{E}f - \frac{1}{t}\sum_{i \in I} f(i) > \epsilon$ happens, then $|\frac{1}{t}\sum_{i \in I} f(i) - \mathbf{E}f| > \epsilon$ happens.

Also as we replace the last d bits of the output of our extractor with the seed, the

samples are distinct according to the construction of Lemma 1.6.2.

According to the definition of averaging sampler, we know that this gives an explicit $(\mu, \alpha\mu, \gamma)$-averaging sampler.

According to the construction described in the proof of Lemma 1.6.2, the output of the sampler is computed by running the extractor following Lemma 1.2.8 for t times in parallel. So the circuit depth is equal to the circuit depth of the extractor Ext.

Let's recall the construction of the Trevisan's extractor Ext.

The encoding procedure is doing the multiplication of the encoding matrix and the input x of length $n' = c\log^a n$. By Lemma 1.2.11, this can be done by a circuit of depth $a+1$.

The last step is the procedure of N-W generator. The selection procedure can be represented as a CNF/DNF, as the seed length for Ext is at most $\Theta(\log n)$. (Detailed proof is the same as the proof of Lemma 1.4.10.)

As a result, we need a circuit of depth $a+2$ to realize Samp.

For the special situation that $c = 0$, the seed length d for Ext can be $\Theta(a \log\log n)$. So $t_0 = 2^d = (\log n)^{\Theta(a)}$. □

By Lemma 1.5.2, we can sample several times to get a block source.

Lemma 1.6.4 (Sample a Block Source)**.** *Let t be any constant in $\mathbb{N}^+$. For any $\delta > 0$, let X be an $(n, k = \delta n)$-source . Let $\mathsf{Samp} : \{0,1\}^r \to [n]^m$ be a (μ_1, μ_2, γ)-averaging sampler where $\mu_1 = (\frac{1}{t}\delta - 2\tau)/\log(1/\tau)$ and $\mu_2 = (\frac{1}{t}\delta - 3\tau)/\log(1/\tau)$, $m = (\frac{t-1}{t}k - \log(1/\epsilon_0))/t$. Let $\epsilon_s = \gamma + 2^{-\Omega(\tau n)}$. For any $i \in [t]$, let U_is be uniform distributions over $\{0,1\}^r$. Let $X_i = X_{\mathsf{Samp}(U_i)}$, for $i \in [t]$.*

It concludes that $\bigcirc_{i=1}^t U_i \circ \bigcirc_{i=1}^t X_i$ is $\epsilon = t(\epsilon_s + \epsilon_0)$-close to $\bigcirc_{i=1}^t U_i \circ \bigcirc_{i=1}^t W_i$

where for every $u \in \mathsf{supp}(\bigcirc_{i=1}^{t} U_i)$*, conditioned on* $\bigcirc_{i=1}^{t} U_i = u$*,* $\bigcirc_{i=1}^{t} W_i$ *is a* $(k_1, k_2, \ldots, k_t)$*-block source with block size* m *and* $k_1 = k_2 = \cdots = k_t = (\delta/t - 3\tau)m$*. Here* ϵ_0 *can be as small as* $1/2^{\Omega(k)}$*.*

Proof. We prove by induction on $i \in [t]$.

If $i = 1$, according to Lemma 1.5.2, we know $U_1 \circ X_1$ is $\epsilon_s = (\gamma + 2^{-\Omega(\tau n)})$-close to $U_1 \circ W$ such that $\forall u \in \mathsf{supp}(U_1), H_\infty(W|_{U_1=u}) = (\delta/t - 3\tau)m$.

Next we prove the induction step.

Suppose $\bigcirc_{j=1}^{i} U_j \circ \bigcirc_{j=1}^{i} X_j$ is $(\epsilon_s + \epsilon_0)i$-close to $\bigcirc_{j=1}^{i} U_j \circ \bigcirc_{j=1}^{i} W_j$, where for every $u \in \{0,1\}^{ir}$, conditioned on $\bigcirc_{j=1}^{i} U_j = u$, $\bigcirc_{j=1}^{i} W_j$ is a $(k_1, k_2, \ldots, k_i)$-block source with block size m and $k_1 = k_2 = \cdots = k_i = (\delta/t - 3\tau)m$.

Consider $i+1$. Recall the Chain Rule Lemma 1.5.13. First notice that $\bigcirc_{j=1}^{i} U_j \circ \bigcirc_{j=1}^{i} X_j \circ X$ has entropy $ir + k$. Then we know that $\bigcirc_{j=1}^{i} U_j \circ \bigcirc_{j=1}^{i} X_j \circ X$ is ϵ_0-close to $\bigcirc_{j=1}^{i} U_j \circ \bigcirc_{j=1}^{i} X_j \circ X'$ such that for every $u \in \{0,1\}^{ir}$ and every $x \in \{0,1\}^{im}$, conditioned on $\bigcirc_{j=1}^{i} U_j = u, \bigcirc_{j=1}^{i} X_j = x$, X' has entropy $k - im - \log(1/\epsilon_0) \geq k/t$ which means the entropy rate is at least δ/t.

By our assumption for i, $\bigcirc_{j=1}^{i} U_j \circ \bigcirc_{j=1}^{i} X_j \circ X'$ is $(\epsilon_s + \epsilon_0)i$-close to $\bigcirc_{j=1}^{i} U_j \circ \bigcirc_{j=1}^{i} W_j \circ \tilde{X}$, where $\tilde{X}$ is a random variable such that $\forall u \in \{0,1\}^{ir}, \forall x \in \{0,1\}^{im}$, $\tilde{X}|_{\bigcirc_{j=1}^{i} U_j = u, \bigcirc_{j=1}^{i} X_j = x}$ has the same distribution as $X'|_{\bigcirc_{j=1}^{i} U_j = u, \bigcirc_{j=1}^{i} W_j = x}$. As a result, for every $u \in \{0,1\}^{ir}$ and $x \in \{0,1\}^{im}$, conditioned on $\bigcirc_{j=1}^{i} U_j = u, \bigcirc_{j=1}^{i} W_j = x$, $\tilde{X}$ has entropy $k - im - \log(1/\epsilon_0) \geq k/t$.

Denote the event $(\bigcirc_{j=1}^{i} U_j = u, \bigcirc_{j=1}^{i} W_j = x)$ as e, by Lemma 1.5.2, by sampling on source $\tilde{X}|_e$, we get $U_{i+1} \circ (\tilde{X}|_e)_{\mathsf{Samp}(U_{i+1})} = U_{i+1} \circ \tilde{X}_{\mathsf{Samp}(U_{i+1})}|_e$. It is ϵ_s-close to $U_{i+1} \circ W|_e$ where $\forall a \in \{0,1\}^r$, $(W|_e)|_{U_{i+1}=a}$ is a $(m, (\delta/t - 3\tau)m)$-source.

Thus $\bigcirc_{j=1}^{i+1} U_j \circ \bigcirc_{j=1}^{i} W_j \circ \tilde{X}_{\mathsf{Samp}(U_{i+1})}$ is ϵ_s-close to $\bigcirc_{j=1}^{i+1} U_j \circ \bigcirc_{j=1}^{i} W_j \circ W$.

Let $W_{i+1} = W$. As a result, $\bigcirc_{j=1}^{i+1} U_j \circ \bigcirc_{j=1}^{i} X_j$ is $(\epsilon_s + \epsilon_0)(i+1)$-close to $\bigcirc_{j=1}^{i+1} U_j \circ \bigcirc_{j=1}^{i+1} W_j$ such that for every $u \in \{0,1\}^{ir}$, conditioned on $\bigcirc_{j=1}^{i+1} U_j = u$, $\bigcirc_{j=1}^{i+1} W_j$ is a $(k_1, k_2, \ldots, k_i)$-block source with block size m and $k_1 = k_2 = \cdots = k_{i+1} = (\delta/t - 3\tau)m$.

This proves that induction step.

□

This lemma reveals a way to get a block source by sampling. Block sources are easier to extract.

1.6.2 Repeating Extraction

Another important technique is the parallel extraction. According to Raz at al. [RRV02], we have the following lemma.

Lemma 1.6.5 ([RRV02])**.** *Let* $\mathsf{Ext}_1 : \{0,1\}^n \times \{0,1\}^{d_1} \to \{0,1\}^{m_1}$ *be a strong* (k,ϵ)*-extractor with entropy loss* Δ_1 *and* $\mathsf{Ext}_2 : \{0,1\}^n \times \{0,1\}^{d_2} \to \{0,1\}^{m_2}$ *be a strong* $(\Delta_1 - s, \epsilon_2)$*-extractor with entropy loss* Δ_2 *for any* $s < \Delta_1$*. Suppose the function* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^{d_1+d_2} \to \{0,1\}^{m_1+m_2}$ *is as follows.*

$$\mathsf{Ext}(x, u_1 \circ u_2) = \mathsf{Ext}_1(x, u_1) \circ \mathsf{Ext}_2(x, u_2)$$

Then Ext *is a strong* $(k, (\frac{1}{1-2^{-s}})\epsilon_1 + \epsilon_2 \leq \epsilon_1 + \epsilon_2 + 2^{-s})$*-extractor with entropy loss* $\Delta_2 + s$*.*

This can be generalized to the parallel extraction for multiple times.

Lemma 1.6.6. *Let X be an (n,k)-source. Let $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a strong (k_0,ϵ)-extractor with $k_0 = k - tm - s$ for any t,s such that $tm + s < k$. Let $\mathsf{Ext}' : \{0,1\}^n \times \{0,1\}^{td} \to \{0,1\}^{tm}$ be constructed as follows.*

$$\mathsf{Ext}'(x, \bigcirc_{i=1}^{t} u_i) = \mathsf{Ext}(x,u_1) \circ \mathsf{Ext}(x,u_2) \circ \cdots \circ \mathsf{Ext}(x,u_t)$$

Then Ext' is a strong $(k, t(\epsilon + 2^{-s}))$-extractor.

Proof. Consider the mathematical induction on j.

For $j = 1$, it is true. As Ext is a strong (k_0,ϵ)-extractor, it is also a strong $(k, j(\epsilon + 2^{-s}))$-extractor.

Next we prove the induction step.

Assume it is true for j. Consider $j+1$.

$$\mathsf{Ext}'(x, \bigcirc_{i=1}^{j+1} u_i) = \mathsf{Ext}'(x, \bigcirc_{i=1}^{j} u_i) \circ \mathsf{Ext}(x, u_{j+1})$$

Here $\mathsf{Ext}'(x, \bigcirc_{i=1}^{j} u_i)$ is a strong $(k, j(\epsilon + 2^{-s}))$-source. Its entropy loss is $k - jm$. Also we know that Ext is a strong $(k - tm - s, \epsilon)$-extractor, thus a strong $(k - jm - s, \epsilon)$-extractor. According to Lemma 1.6.5, $\mathsf{Ext}'(x, \bigcirc_{i=1}^{j+1} u_i)$ is a strong $(k, (j+1)(\epsilon + 2^{-s}))$-extractor. Its entropy loss is $k - (j+1)m$.

This completes the proof.

□

Lemma 1.6.6 shows a way to extract more bits. Assume we have an (n,k)-source and an extractor, if the output length of the extractor is $k^\beta, \beta < 1$, then we can extract several times to get a longer output. However, if we merely do it in this way, we need

a longer seed. In fact, if we extract enough times to make the output length to be $\Theta(k)$, we need a seed with length $\Theta(k^{1-\beta}\log n)$. This immediately gives us the following theorem.

Theorem 1.6.7. *For any constant $a, c \in \mathbb{N}$, $\gamma \in (0,1)$, any $k = \Theta(n/\log^c n)$, $\epsilon = 1/2^{\Theta(\log^a n)}$, there exists an explicit strong (k,ϵ)-extractor $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ in $\mathcal{AC}^0$ of depth $3a+c+7$. The locality is $O(\log^{2a+c+4} n)$. The seed length $d = O(\frac{k\log(1/\epsilon)}{n^{1/10800}\log n})$. The output length $m = (1-\gamma)k$.*

Proof. Let $\mathsf{Ext}_0 : \{0,1\}^n \times \{0,1\}^{d_0} \to \{0,1\}^{m_0}$ be a $(k_0, \epsilon_0 = \epsilon/n)$ extractor following from Theorem 1.5.18. Here $k_0 = k - tm_0 - s$ where $s = \log(n/\epsilon)$, $t = (1-\gamma)k/m_0$. By lemma 1.6.6, we know that there exists a (k,ϵ') extractor Ext with $\epsilon' = t(\epsilon_0 + 2^{-s}) \leq \epsilon$. The output length is $(1-\gamma)k$.

According to the construction in Lemma 1.6.6, Ext has the same circuit depth and locality as Ext_0. The seed length is $t \times d_0 = O(\frac{k\log(1/\epsilon)}{n^{1/10800}\log n})$.

□

If we only consider local extractors then similarly we have the following.

Theorem 1.6.8. *For any constant $c \in \mathbb{N}$, $\gamma \in (0,1)$, any $k = \Theta(\frac{n}{\log^c n})$, there exists an explicit construction of a strong (k,ϵ)-extractor $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, where ϵ can be as small as $2^{-k^{\Omega(1)}}$, $d = O(\frac{k\log(1/\epsilon)}{n^{1/10800}\log n})$, $m = (1-\gamma)k$ and the locality is $\log^2(1/\epsilon)\log^{c+O(1)} n$.*

1.6.3 The Construction

In order to extract more bits while keeping seed length small, we use classic bootstrapping techniques. Our construction is still in $\mathcal{AC}^0$ but it does not have small locality

.

Construction 1.6.9. *For any constant $a, c \in \mathbb{N}$, any $k = \delta n = \Theta(n/\log^c n), \epsilon = 1/2^{\Theta(\log^a n)}$, we construct a (k, ϵ)-extractor $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ where $d = O(\log n + \frac{\log(n/\epsilon)\log(1/\epsilon)}{\log n})$, $m = O(\delta k)$.*

- *Let X be an $(n, k = \delta n)$-source*
- *Let $t \geq 10800$ be a large enough constant.*
- *Let $\mathsf{Samp} : \{0,1\}^r \to [n]^{m_s}$ be a (μ_1, μ_2, γ)-averaging sampler following from Lemma 1.6.3, where $\mu_1 = (\frac{1}{t}\delta - 2\tau)/\log(1/\tau)$ and $\mu_2 = (\frac{1}{t}\delta - 3\tau)/\log(1/\tau)$, $m_s = (\frac{t-1}{t}k - \log(1/\epsilon_0))/t$, $\tau = \frac{1}{4}\delta$, $\gamma = \epsilon/n$. Let $\epsilon_s = \gamma + 2^{-\Omega(\tau n)}$.*
- *Let $\mathsf{Ext}_0 : \{0,1\}^{n_0 = m_s} \times \{0,1\}^{d_0} \to \{0,1\}^{m_0}$ be a (k_0, ϵ_0)-extractor following from Theorem 1.5.18 where $k_0 = 0.1(\frac{1}{t}\delta - 3\tau)m_s - s$, $\epsilon_0 = \epsilon/(10tn)$, $d_0 = O(\log n_0 + \frac{\log(n_0/\epsilon_0)\log(1/\epsilon_0)}{\log n_0})$, $m_0 = n_0^{1/10800}\Theta(\log(n_0/\epsilon_0))$. Let s be such that $2^{-s} \leq \epsilon/(10tn)$.*

Next we construct the function Ext as follows.

1. *Get Let $X_i = X_{\mathsf{Samp}(X, S_i)}$ for $i \in [t]$, where $S_i, i \in [t]$ are independent uniform distributions.*
2. *Get $Y_t = \mathsf{Ext}_0(X_t, U_0)$ where U_0 is the uniform distribution with length d_0.*
3. *For $i = t - 1$ to 1, get $Y_i = \mathsf{Ext}'(X_i, Y_{i+1})$ sequentially. The function Ext' is defined as follows.*

$$\mathsf{Ext}'(x, r) = \bigcirc_{i=1}^{\min\{\lfloor |r|/d_0 \rfloor, \lfloor 0.9(\frac{1}{t}\delta - 3\tau)m_s/m_0 \rfloor\}} \mathsf{Ext}_0(x, r_i)$$

where $r = \bigcirc_{i=1}^{\lfloor |r|/d_0 \rfloor} r_i \circ r'$ *for some extra bits* r' *and* $\forall i, |r_i| = d_0$.

4. *Output* $\mathsf{Ext}(X, U_d) = Y_1 = \mathsf{Ext}'(X_1, Y_2)$, *where* $U_d = U_0 \circ \bigcirc_{i=1}^{t} S_i$.

Lemma 1.6.10. *For* $\epsilon_1 = 1/2^{\Omega(k)}$, $\bigcirc_{i=1}^{t} S_i \circ \bigcirc_{i=1}^{t} X_i$ *is* $t(\epsilon_s + \epsilon_1)$*-close to* $\bigcirc_{i=1}^{t} S_i \circ \bigcirc_{i=1}^{t} W_i$.

Here S_i*s are independent uniform distributions and* $\forall r \in \mathsf{supp}(\bigcirc_{i=1}^{t} S_i)$*, conditioned on* $\bigcirc_{i=1}^{t} S_i = r$, $\bigcirc_{i=1}^{t} W_i$ *is a* $(k_1, k_2, \ldots, k_t)$*-block source with* $k_1 = k_2 = \cdots = k_t = k' = (\frac{1}{t}\delta - 3\tau)m_s$.

Proof. It follows from Lemma 1.6.4.

□

Lemma 1.6.11. *In Construction 1.6.9, the function* Ext *is a strong* (k, ϵ)*-extractor.*

Proof of Lemma 1.6.11. By Lemma 1.6.10, $\bigcirc_{i=1}^{t} S_i \circ \bigcirc_{i=1}^{t} X_i$ is $t(\epsilon_s + \epsilon_1) = 1/\mathsf{poly}(n)$-close to $\bigcirc_{i=1}^{t} S_i \circ B$ where $B = B_1 \circ B_2 \circ \ldots \circ B_t$. The S_is are independent uniform distributions. Also $\forall s \in \mathsf{supp}(\bigcirc_{i=1}^{t} S_i)$, conditioned on $\bigcirc_{i=1}^{t} S_i = s$, B is a $(k_1, k_2, \ldots, k_t)$-block source with $k_1 = k_2 = \cdots = k_t = k' = (\frac{1}{t}\delta - 3\tau)m_s$. We denote the first i blocks to be $\tilde{B}_i = \bigcirc_{j=1}^{i} B_i$.

Let $Y'_i = \mathsf{Ext}'(B_i, Y'_{i+1})$ for $i = 1, 2, \ldots, t$ where $Y'_{t+1} = U_0$ is the uniform distribution with length d_0.

Next we use induction over i (from t to 1) to show that

$$\mathsf{SD}(U_0 \circ Y'_i, U) \leq (t + 1 - i)k(\epsilon_0 + 2^{-s}).$$

The basic step is to prove that $\forall b_1, b_2, \ldots, b_{t-1} \in \{0,1\}^{m_s}$, conditioned on $B_1 = b_1, \ldots, B_{t-1} = b_{t-1}$, $\mathsf{SD}(U_0 \circ Y'_t, U) \leq k(\epsilon_0 + 2^{-s})$. According to the

definition of Ext',

$$\mathsf{SD}(U_0 \circ \mathsf{Ext}'(B_t, U_0), U) \leq \epsilon_0.$$

This proves the basic step.

For the induction step, assume that $\forall b_1, b_2, \ldots, b_{i-1} \in \{0,1\}^{m_s}$, conditioned on $B_1 = b_1, \ldots, B_{i-1} = b_{i-1}$,

$$\mathsf{SD}(U_0 \circ Y'_i, U) \leq (t+1-i)k(\epsilon_0 + 2^{-s}).$$

Consider $U_0 \circ Y'_{i-1} = U_0 \circ \mathsf{Ext}'(B_{i-1}, Y'_i)$.

We know that $\forall b_1, b_2, \ldots, b_{t-2} \in \{0,1\}^{m_s}$, conditioned on $B_1 = b_1, \ldots, B_{i-2} = b_{i-2}$, $\tilde{B}_{i-1} \circ U_0 \circ Y'_i$ is a convex combination of $b_{i-1} \circ U_0 \circ Y'_i, \forall b_{i-1} \in \mathsf{supp}(\tilde{B}_{i-1})$. As a result,

$$\mathsf{SD}(\tilde{B}_{i-1} \circ U_0 \circ Y'_i, \tilde{B}_{i-1} \circ U) \leq (t+1-i)k(\epsilon_0 + 2^{-s}).$$

Thus, $\forall b_1, b_2, \ldots, b_{t-2} \in \{0,1\}^{m_s}$, conditioned on $B_1 = b_1, \ldots, B_{i-2} = b_{i-2}$, as $\tilde{B}_{i-1} \circ U_0 \circ Y'_i$ is a convex combination of $b_{i-1} \circ U_0 \circ Y'_i, \forall b_{i-1} \in \mathsf{supp}(\tilde{B}_{i-1})$ and $\tilde{B}_{i-1} \circ U$ is a convex combination of $b_{i-1} \circ U, \forall b \in \mathsf{supp}(\tilde{B}_{i-1})$, by Lemma 1.2.4 part 2,

$$\begin{aligned} &\mathsf{SD}(U_0 \circ \mathsf{Ext}'(B_{i-1}, Y'_i), U_1 \circ \mathsf{Ext}'(B_{i-1}, U_2)) \\ \leq &\mathsf{SD}(\tilde{B}_{i-1} \circ U_0 \circ Y'_i, \tilde{B}_{i-1} \circ U) \qquad (1.18) \\ \leq &(t+1-i)k(\epsilon_0 + 2^{-s}). \end{aligned}$$

Here $U = U_1 \circ U_2$. U_1 is the uniform distribution having $|U_1| = |U_0|$. U_2 is the uniform distribution having $|U_2| = |Y'_i|$.

According to the definition of Ext' and Lemma 1.6.6, we know that $\forall b_1, b_2, \ldots, b_{i-2} \in \{0,1\}^{m_s}$, conditioned on $B_1 = b_1, \ldots, B_{i-2} = b_{i-2}$,

$$\mathsf{SD}(U_1 \circ \mathsf{Ext}'(B_{i-1}, U_2), U) \leq k(\epsilon_0 + 2^{-s}).$$

So according to triangle inequality of Lemma 1.2.4, $\forall b_1, b_2, \ldots, b_{t-2} \in \{0,1\}^{m_s}$, conditioned on $B_1 = b_1, \ldots, B_{i-2} = b_{i-2}$,

$$\begin{aligned}
&\mathsf{SD}(U_0 \circ Y'_{i-1}, U) \\
=&\mathsf{SD}(U_0 \circ \mathsf{Ext}'(B_{i-1}, Y'_i), U) \\
\leq&\mathsf{SD}(U_0 \circ \mathsf{Ext}'(B_{i-1}, Y'_i), U_1 \circ \mathsf{Ext}'(B_{i-1}, U_2)) + \mathsf{SD}(U_1 \circ \mathsf{Ext}'(B_{i-1}, U_2), U) \\
\leq&(t+1-i)k(\epsilon_0 + 2^{-s}) + k(\epsilon_0 + 2^{-s}) \\
=&(t+1-(i-1))k(\epsilon_0 + 2^{-s}).
\end{aligned} \tag{1.19}$$

This proves the induction step.

So we have $\mathsf{SD}(U_0 \circ Y'_1, U) \leq tk(\epsilon_0 + 2^{-s})$.

As a result,

$$\begin{aligned}
&\mathsf{SD}(U_d \circ \mathsf{Ext}(X, U_d), U) \\
=&\mathsf{SD}(U_0 \circ \bigcirc_{i=1}^{t} S_i \circ Y_1, U) \\
\leq&\mathsf{SD}(U_0 \circ \bigcirc_{i=1}^{t} S_i \circ Y_1, U_0 \circ \bigcirc_{i=1}^{t} S_i \circ Y_1') + \mathsf{SD}(U_0 \circ \bigcirc_{i=1}^{t} S_i \circ Y_1', U) \\
\leq&\mathsf{SD}(U_0 \circ \bigcirc_{i=1}^{t} S_i \circ \bigcirc_{i=1}^{t} X_i, U_0 \circ \bigcirc_{i=1}^{t} S_i \circ \bigcirc_{i=1}^{t} B_i) + \mathsf{SD}(U_0 \circ \bigcirc_{i=1}^{t} S_i \circ Y_1', U) \\
\leq&t(\epsilon_s + \epsilon_1) + tk(\epsilon_0 + 2^{-s}).
\end{aligned} \tag{1.20}$$

According to the settings of ϵ_0, ϵ_s, t and by setting ϵ_1 to be small enough, we know the error is at most ϵ.

□

Lemma 1.6.12. *In Construction 1.6.9, the length of Y_i is*

$$|Y_i| = \Theta(\min\{m_0(\frac{m_0}{d_0})^{t-i}, 0.9(\frac{1}{t}\delta - 3\tau)m_s\}).$$

Specifically, $m = |Y_1| = \Theta((\frac{1}{t}\delta - 3\tau)m_s) = \Theta(\delta k)$.

Proof. For each time we compute $Y_i = \mathsf{Ext}'(X_i, Y_{i+1})$, we know $|Y_i| \leq |Y_{i+1}|(\frac{m_0}{d_0})$. Also according to the definition of Ext', $|Y_i| \leq 0.9(\frac{1}{t}\delta - 3\tau)m_s$. So

$$|Y_i| = \Theta(\min\{m_0(\frac{m_0}{d_0})^{t-i}, 0.9(\frac{1}{t}\delta - 3\tau)m_s\})$$

for $i \in [t]$.

By Theorem 1.5.18, $m_0 = \Theta(n_0^{1/10800}\log n)$. Also we know that $n_0 = m_s = O(tk)$. As a result, when $t \geq 10800$, $m_0(\frac{m_0}{d_0})^{t-1} = \omega(m_s)$. As a result, $m = |Y_1| =$

$\Theta((\frac{1}{t}\delta - 3\tau)m_s) = \Theta(\delta k)$.

□

Lemma 1.6.13. *In Construction 1.6.9, the seed length* $d = O(\log n + \frac{\log(n/\epsilon)\log(1/\epsilon)}{\log n})$.

Proof. The seed for this extractor is $U_d = U_0 \circ \bigcirc_{i=1}^{t} S_i$. So $|U_d| = |U_0| + \Sigma_i^t |S_i| = O(\log n + \frac{\log(n/\epsilon)\log(1/\epsilon)}{\log n}) + O(\log(1/\epsilon)) = O(\log n + \frac{\log(n/\epsilon)\log(1/\epsilon)}{\log n})$.

□

Lemma 1.6.14. *In Construction 1.6.9, the function* Ext *is in* $\mathcal{AC}^0$. *The depth of the circuit is* $O(a + c + 1)$.

Proof. We in fact run Samp and Ext_0 for constant number of times in sequential. So the total depth is $O(a + c)$ as the depth of Samp and Ext_0 are both $O(a + c + 1)$.

□

Theorem 1.6.15. *For any constant* $a, c \in \mathbb{N}$, *any* $k = \delta n = \Theta(n/\log^c n)$, $\epsilon = 1/2^{\Theta(\log^a n)}$, *there exists an explicit strong* (k, ϵ)*-extractor* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *in* $\mathcal{AC}^0$ *with depth* $O(a + c + 1)$, *where* $d = O(\log n + \frac{\log(n/\epsilon)\log(1/\epsilon)}{\log n})$, $m = \Omega(\delta k)$.

Proof. By Construction 1.6.9, Lemma 1.6.11, Lemma 1.6.12, Lemma 1.6.13 , and Lemma 1.6.14 , the conclusion immediately follows.

□

Theorem 1.6.16. *For any constant* $\gamma \in (0,1)$, $a, c \in \mathbb{N}$, *any* $k = \delta n = \Theta(n/\log^c n)$, $\epsilon = 1/2^{\Theta(\log^a n)}$, *there exists an explicit strong* (k, ϵ)*-extractor* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *in* $\mathcal{AC}^0$ *with depth* $O(a + c + 1)$ *where* $d = O((\log n + \frac{\log(n/\epsilon)\log(1/\epsilon)}{\log n})/\delta)$, $m = (1 - \gamma)k$.

Proof. Let the extractor following Theorem 1.6.15 be $\mathsf{Ext}_0 : \{0,1\}^{n_0} \times \{0,1\}^{d_0} \to \{0,1\}^{m_0}$ which is a (k_0, ϵ_0)-extractor with $n_0 = n, k_0 = \gamma k - s$ for some $s < \gamma k$. The construction of Ext is

$$\mathsf{Ext}(x,u) = \bigcirc_{i=1}^{t} \mathsf{Ext}_0(x,u_i).$$

Here t is such that $tm_0 = (1-\gamma)k$.

By Lemma 1.6.15, we know that $m_0 = \Theta(\delta k)$ where $\delta = \Theta(\frac{1}{\log^c n})$. So $t = \Theta(1/\delta)$. By Lemma 1.6.6, if $tm_0 = (1-\gamma)k$, then Ext is a (k,ϵ)-extractor with output length $(1-\gamma)k$ and error $t(\epsilon_0 + 2^{-s})$.

We choose s to be large enough and ϵ_0 to be small enough such that the error is at most ϵ. The seed length $d = td_0$. By Theorem 1.6.15, $d_0 = O(\log n + \frac{\log(n/\epsilon)\log(1/\epsilon)}{\log n})$, $m_0 = \Omega(\delta k)$, so $d = O((\log n + \frac{\log(n/\epsilon)\log(1/\epsilon)}{\log n})/\delta)$, $m = (1-\gamma)k$. The circuit depth maintains the same as that in Theorem 1.6.15 because the extraction is conducted in parallel. □

1.7 Deterministic Extractor for Bit-fixing Source

We use two crucial tools. One is the extractor for non-oblivious bit-fixing sources, proposed by Chattopadhyay and Zuckerman [CZ16] and improved by Li [Li16].

Theorem 1.7.1 ([Li16] Theorem 1.11)**.** *Let c be a constant. For any $\beta > 0$ and all $n \in \mathbb{N}$, there exists an explicit extractor $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}^m$ such that for any (q,t,γ)-non-oblivious bit-fixing source X on n bits with $q \le n^{1-\beta}$, $t \ge c\log^{21} n$ and $\gamma \le 1/n^{t+1}$,*

$$\mathsf{SD}(\mathsf{Ext}(X), U) \le \epsilon$$

where $m = t^{\Omega(1)}$, $\epsilon = n^{-\Omega(1)}$.

The extractor can be computed by standard circuits of depth $\lceil \frac{\log m}{\log \log n} \rceil + O(1)$.

To see the depth is $\lceil \frac{\log m}{\log \log n} \rceil + O(1)$, let's briefly recall the construction of the extractor. It first divides the input in to $n^{O(1)}$ blocks. Then for each block, it applys the extractor from [Li16] Theorem 4.1 which has depth 4. At last, it conducts a multiplication between a matrix of size $m \times O(m)$ and a vector of dimension $O(m)$, both over $\mathbb{F}_2$. The last step can be computed by a circuit of depth $\lceil \frac{\log m}{\log \log n} \rceil + 1$ by Lemma 1.2.11 part 2.

The other tool is the design extractor introduced by Li [Li12].

Definition 1.7.2 ([Li12])**.** *An* $(N, M, K, D, \alpha, \epsilon)$*-design extractor is a bipartite graph with left vertex set* $[N]$*, right vertex set* $[M]$*, left degree* D *such that the following properties hold.*

- *(extractor property) For any subset* $S \subseteq [M]$*, let* $\rho_S = |S|/M$*. For any vertex* $v \in [N]$*, let* $\rho_v = |\Gamma(v) \cap S|/D$*. Let* $Bad_S = \{v \in [N] : |\rho_v - \rho_S| > \epsilon\}$*, then* $|Bad_S| \leq K$*. (*$\Gamma(\cdot)$ *outputs the set of all neighbors of the input.)*
- *(design property) For any two different vertices* $u, v \in [N]$*,* $|\Gamma(u) \cap \Gamma(v)| \leq \alpha D$*.*

Construction 1.7.3. *For any constant* $a \in \mathbb{N}$*, any* $t = \mathsf{poly}(\log n)$*, the deterministic extractor* $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}^{m=t^{\Omega(1)}}$ *for any* $(n, \delta n = \Theta(n/\log^a n))$*-bit-fixing source is constructed as the follows.*

- *Construct an* $(N, M, K, D, \alpha, \epsilon)$*-design extractor, where* $M = n, K = n^{1/0.9}, N = n^{1/0.3}$*,* $\epsilon = 1/\log^c N$*,* $D = \log^b N$*,* $\alpha = D/M + \epsilon$*, for* $c = \lceil \log t / \log \log N \rceil + a + 1$ *and large enough constant* $b = \Theta(c)$*.*

- *Let* $Y = (Y_1, Y_2, \ldots, Y_N)$. *Compute* $Y_i = \bigoplus_{j \in \Gamma(i)} X_j$, *for* $i = 1, \ldots, N$, *by taking* i *as the* i*th vertex in the left set of the design extractor.*

- *Let* $\mathsf{Ext}(X) = \mathsf{Ext}'(Y)$ *where* $\mathsf{Ext}' : \{0,1\}^N \to \{0,1\}^m$ *is the extractor from Theorem 1.7.1 with error* $\epsilon = n^{-\Omega(1)}$.

Lemma 1.7.4. *An* $(N, M, K, D, \alpha, \epsilon)$*-design extractor, where* $K = N^{1/3}, M = K^{0.9}$, $\epsilon = 1/\log^c N$, $D = \log^b N$,$\alpha = D/M + \epsilon$, *for any constant* c *and large enough constant* $b = \Theta(c)$, *can be constructed in polynomial time.*

Proof. In [Li12] it is showed that design extractors can be constructed in deterministic polynomial time by a greedy algorithm.

The construction is based on a (k_0, ϵ)-extractor $\mathsf{Ext}_0 : \{0,1\}^{n_0} \times \{0,1\}^{d_0} \to \{0,1\}^{m_0}$, from Theorem 1.2.7 (almost optimal parameters), for any (n_0, k_0)-source, where $n_0 = 4k_0, m_0 = 0.9k_0, d_0 = O(\log(n_0/\epsilon))$. Also we substitute the first d_0 bits of the output by the seed s.t. every left vertex has exactly 2^{d_0} neighbors. Recall the greedy algorithm proposed by Li [Li12], which picks vertices one by one, deleting the vertices which does not meet the design property before each picking. At last we can get $2^{n_0-k_0} \geq 2^{3k_0}$ left vertices. Let $N = 2^{3k_0}, K = 2^{k_0}, M = 2^{m_0}$. We know that $\epsilon = 1/\log^c N$ and $d_0 = O(\log(n_0/\epsilon))$. Thus if $b = \Theta(c)$ is large enough, D can be $\log^b N$ by adding extra random bits to adjust the length of the seed.

□

Lemma 1.7.5. *For any constant* $a \in \mathbb{N}$, *any* $t = \mathsf{poly}(\log n)$, *if* X *is an* $(M, \delta M = \Theta(M/\log^a M))$*-bit-fixing source, then* $Y = g(X)$ *is a* $(q, t, 0)$*-non-oblivious bit-fixing source, where* $q = K$.

Proof. Assume the coordinates of random bits of X form the set S. By the extractor property of design extractors, the number of left vertex x, such that $|\rho_x - \rho_S| > \epsilon$, is at most K. These vertices form the set Bad_S.

We prove that for any subset $V \subseteq [N] \backslash Bad_S$ with size $|V| \leq t$, $\bigoplus_{j \in V} Y_j$ is uniformly distributed.

Let $V = \{v_1, v_2, \ldots, v_{t'}\}$ be a subset of $[N] \backslash Bad_S$, where $t' \leq t$. So $|\Gamma(v_{t'}) \cap S| \geq (\delta - \epsilon)D$. By the design property of design extractors, for any $i = 1, 2, \ldots, t' - 1$, $|\Gamma(v_{t'}) \cap \Gamma(v_i)| \leq \alpha D$. So $|(\Gamma(v_{t'}) \cap S) \backslash \bigcup_{i=1}^{t'-1} \Gamma(v_i)| \geq (\delta - \epsilon)D - t \cdot \alpha D \geq 1$ for $c = \lceil \log t / \log \log N \rceil + a + 1$ and large enough constant b. Thus $\bigoplus_{j \in V} Y_j$ is uniformly distributed because some uniform random bits in $\Gamma(v_{t'}) \cap S$ cannot be canceled out by bits in $\bigcup_{i=1}^{t'-1} \Gamma(v_i)$.

By the Information Theoretic XOR-Lemma in [Gol95], $Y_{[N] \backslash Bad_S}$ is t-wise independent. Thus $Y = g(X)$ is a $(q = K, t, 0)$-non-oblivious bit-fixing source.

□

Theorem 1.7.6. *For any constant $a \in \mathbb{N}$, there exists an explicit deterministic $(k = \delta n = \Theta(n/\log^a n), \epsilon = n^{-\Omega(1)})$-extractor $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}^m$ that can be computed by $\mathcal{AC}^0$ circuits of depth $\Theta(\frac{\log m}{\log \log n} + a)$, for any (n, k)-bit-fixing source, where m can be any $\mathsf{poly}(\log n)$.*

Proof. We claim that Construction 1.7.3 gives the desired extractor. Let X be an (n, k)-bit-fixing source. By lemma 1.7.5, we get Y which is a $(q, t, 0)$-non-oblivious bit-fixing source with length $\Theta(n^{1/0.3})$, where $q = \Theta(n^{1/0.9})$ and t can be any large enough $\mathsf{poly}(\log n)$.

By Theorem 1.7.1, $\mathsf{Ext}(X) = \mathsf{Ext}'(Y)$ is ϵ-close to uniform. The output length

$m = t^{\Omega(1)}$ can be any $\mathrm{poly}(\log n)$ as we can set t to be any $\mathrm{poly}(\log n)$.

We show that the circuit for computing the extractor is in uniform $\mathcal{AC}^0$. In Construction 1.7.3, each Y_i is the XOR of poly-logarithmic bits of X. Also the extractor of Theorem 1.7.1 is in $\mathcal{AC}^0$. So the overall construction is in $\mathcal{AC}^0$. It is in uniform $\mathcal{AC}^0$ because the design extractor can be constructed in polynomial time by Lemma 1.7.4 while all other operations are explicit and can be computed by uniform $\mathcal{AC}^0$ circuits.

The depth of the circuit is $\Theta(\frac{\log m}{\log\log n} + a)$. Because in Construction 1.7.3, $c = \lceil \log t / \log\log N \rceil + a + 1$ and $b = \Theta(c)$. By lemma 1.2.11 part 2, the XOR of $D = 1/\log^b N$ bits can be computed by circuits of depth b. Also the depth of Ext' is $\lceil \frac{\log m}{\log\log n} \rceil + O(1)$. Thus the overall depth is $\Theta(\frac{\log m}{\log\log n} + a)$.

□

Next we do error reduction. Our method is based on the XOR lemma given by Barak, Impagliazzo and Wigderson [BIW06].

Lemma 1.7.7 ([BIW06] Lemma 3.15)**.** *Let $Y_1, Y_2, \ldots, Y_t$ be independent distributions over $\mathbb{F}$ such that $\forall i \in [t], \mathsf{SD}(Y_i, U) \leq \epsilon$. Then*

$$\mathsf{SD}(\sum_{i=1}^{t} Y_i, U) \leq (2\epsilon)^t,$$

where U is uniform over $\mathbb{F}$.

Proof. For simplicity, let $\mathbb{F} = \{0, 1, \ldots, M-1\}$.

We use induction to show that for $j = 1, 2, \ldots, t$, $\mathsf{SD}(\sum_{i=1}^{j} Y_i, U) \leq (2\epsilon)^j$.

As Y_1 is ϵ-close to uniform, this shows the base case.

Let $Y' = \sum_{i=1}^{j-1} Y_i$. Suppose $\mathsf{SD}(Y', U) \leq (2\epsilon)^{j-1}$.

Let $p' = (p'_0, p'_1, \ldots, p'_{M-1})$ be such that $p'_i = \Pr[Y' = i_b] = 1/M + \delta'_i$, for $i = 0, 1, \ldots, M-1$, where i_b is the binary form of i.

We know that $\mathsf{SD}(Y', U) = 1/2(\sum_{i=1}^{M-1} |\delta'_i|)$ and $\sum_{i=0}^{M-1} \delta'_i = 0$.

Let $p = (p_0, p_1, \ldots, p_{M-1})$ be such that $p_i = \Pr[Y_j = i_b] = 1/M + \delta_i$ for $i = 0, 1, \ldots, M-1$. We know that $\mathsf{SD}(Y_j, U) = 1/2(\sum_{i=1}^{M-1} |\delta_i|)$ and $\sum_{i=0}^{M-1} \delta_i = 0$.

So

$$\begin{aligned}
\Pr[Y' + Y_j = i_b]| &= \sum_{k=0}^{M-1} \Pr[Y' = k_b] \Pr[Y_j = (i-k)_b] \\
&= \sum_{k=0}^{M-1} p'_k \cdot p_{i-k} \\
&= 1/M + 2(\sum_{k=0}^{M-1} \delta_k)/M + \sum_{k=0}^{M-1} \delta_k \delta_{i-k} \\
&= 1/M + \sum_{k=0}^{M-1} \delta_k \delta_{i-k}.
\end{aligned} \tag{1.21}$$

Thus $|\Pr[Y' + Y_j = i_b] - \Pr[U = i_b]| = |\sum_{k=0}^{M-1} \delta_k \delta_{i-k}|$.

As a result,

$$\begin{aligned}
\mathsf{SD}(Y' + Y_j, U) &= 1/2 \sum_{i=0}^{M-1} |\Pr[Y' \oplus Y_j = i_b] - \Pr[U = i_b]| \\
&= 1/2 \sum_{i=0}^{M-1} |\sum_{k=1}^{M-1} \delta_k \delta_{i-k}| \\
&\leq 1/2 (\sum_{k=0}^{M-1} \sum_{l=0}^{M-1} |\delta_k \delta_l|) \\
&= 1/2 (\sum_{i=1}^{M-1} |\delta'_i|)(\sum_{i=1}^{M-1} |\delta_i|) \\
&\leq (2\epsilon)^j.
\end{aligned} \tag{1.22}$$

□

Theorem 1.7.8. *If* $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}^m$ *is a* (k, ϵ)*-extractor for* (n, k)*-bit-fixing sources, then for every* $l \in \mathbb{N}$*, the function* $\mathsf{Ext}' : \{0,1\}^{ln} \to \{0,1\}^m$*, given by* $\mathsf{Ext}'(x_1, \ldots, x_l) = \oplus_{i \in [l]} \mathsf{Ext}(x_i)$ *is a* $(2lk, \epsilon^{\Theta(l)})$*-extractor for* $(ln, 2lk)$*-bit-fixing sources.*

Proof. Let $X = (X^{(1)}, \ldots, X^{(l)})$ be an $(ln, 2lk)$-bit-fixing source. Then for $\delta = k/n$ fraction of $j \in [l]$, $X^{(j)}$ is an $(n, \delta n)$-bit-fixing source. Because if not, the total number of random bits is at most $\delta l \cdot n + (1 - \delta) l \delta n < 2\delta l n = 2lk$. Also we know that $X^{(1)}, X^{(2)}, \ldots, X^{(l)}$ are independent because X is a bit-fixing source. We regard each $\mathsf{Ext}(X^{(i)})$ as a random element (coefficients of the corresponding polynomial) in $\mathbb{F}_{2^m}$. By Lemma 1.7.7, $\mathsf{Ext}'(X)$ is $\epsilon^{\Theta(l)}$-close to uniform.

□

By using the deterministic extractor in Theorem 1.7.6, combining with Theorem

1.7.8, adjusting the parameters, we get the following result.

Theorem 1.7.9. *For any constant $a, c \in \mathbb{N}$, there exists an explicit deterministic $(k = \Theta(n/\log^a n), \epsilon = 2^{-\log^c n})$-extractor $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}^m$ that can be computed by $\mathcal{AC}^0$ circuits of depth $\Theta(\frac{\log m}{\log\log n} + a + c)$, for any (n,k)-bit-fixing sources, where m can be any $\mathsf{poly}\log n$.*

Finally we do output length optimization by applying the same technique as that in [GVW15]. The technique is given by Gabizon et al.[GRS04].

Theorem 1.7.10. *For any constant $a, c \in \mathbb{N}$ and any constant $\gamma \in (0,1]$, there exists an explicit deterministic $(k = \Theta(n/\log^a n), \epsilon = 2^{-\log^c n})$-extractor $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}^{(1-\gamma)k}$ that can be computed by $\mathcal{AC}^0$ circuits of depth $\Theta(a+c+1)$, for any (n,k)-bit-fixing sources.*

proof sketch. The difference between our construction and [GVW15] Theorem 5.12 is that, for the three crucial components in the construction, we use the deterministic extractor of Theorem 1.7.9, the seeded extractor of Theorem 1.6.16 and the averaging sampler in Lemma 1.5.3 instead. We briefly describe the construction as the follows.

- A deterministic ϵ_1-error extractor $\mathsf{Ext}_1 : \{0,1\}^n \to \{0,1\}^{r+r_2}$ for $(n, \mu's)$-bit-fixing sources, by Theorem 1.7.9;
- A seeded ϵ_2-error extractor $\mathsf{Ext}_2 : \{0,1\}^n \times \{0,1\}^{r_2} \to \{0,1\}^m$ for $(n, \mu n - s)$-bit-fixing sources, by Theorem 1.6.16;
- An (μ, μ', θ)-averaging sampler $\mathsf{Samp} : \{0,1\}^r \to [n]^s$, by Lemma 1.6.3 .

We set $\mu = k/n, \mu' = \mu/2, s = k/2$ such that $\mu' s = \Theta(n/\log^{2a} n)$ and $\mu n - s = \Theta(n/\log^{a} n)$. Also we set $\epsilon_1 = 2^{-\log^{3c} n}$, $\epsilon_2 = \theta = 2^{-\log^{2c} n}$, $m = (1-\gamma)k$, $r_2 = (\log n)^{\Theta(a+c)}$ and $r = \Theta(\log^{2c} n)$.

Theorem 7.1 of [GRS04] constructs a deterministic extractor $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}^m$ for $(n, \mu n)$-bit-fixing sources, where the error is $\epsilon = \epsilon_2 + 2^{r+3}\epsilon_1 + 3\theta$. So $\epsilon \leq 2^{-\log^{c} n}$. The construction is $\mathsf{Ext}(x) = \mathsf{Ext}_2(x_{[n]\setminus S(Z_1)} \circ 0^t, Z_2)$, where Z_1 is the first r bits of $\mathsf{Ext}_1(X)$ and Z_2 is the last r_2 bits of $\mathsf{Ext}_1(X)$.

Here we only need to compute the depth of the circuit. We know that $r + r_2 = (\log n)^{\Theta(a+c)}$. The depth of the final extractor is the sum of the depths of all three components. By Theorem 1.7.9, the depth for the deterministic extractor is $\Theta(a + c + 1)$. By theorem 1.6.16, the depth for the seeded extractor is also $\Theta(a + c + 1)$. By Lemma 1.6.3, the depth for the sampler is $\Theta(c + 1)$. So the overall depth is $\Theta(a + c + 1)$.

□

1.8 Applications

In this section, we give some constructions of PRGs based on our AC0 extractor.

1.8.1 PRG in AC0 Based on Random Local One-way Function

Our first construction is based on random local one-way functions following the method of Applebaum [App13].

Let $\mathsf{dist}(\cdot)$ denotes the hamming distance between the two input strings (with equal length).

Definition 1.8.1 (Hypergraphs [App13])**.** *An (n, m, d) hypergraph is a graph over n vertices and m hyperedges each of cardinality d. For each hyperedge $S = (i_0, i_1, \ldots, i_{d-1})$, the indices $i_0, i_1, \ldots, i_{d-1}$ are ordered. The hyperedges of G are also ordered. Let G be denoted as $([n], S_0, S_1, \ldots, S_{m-1})$ where for $i = 0, 1, \ldots, m-1$, S_i is a hyperedge.*

Remark 1.8.2. *Here we do not require the indices $i_0, i_1, \ldots, i_{d-1}$ to be distinct. This setting is the same as that in [BQ12] and [Gol11] (Random Construction).*

Definition 1.8.3 (Predicate)**.** *A d-ary predicate $Q : \{0,1\}^d \to \{0,1\}$ is a function which partitions $\{0,1\}^d$ in to V_0 and V_1, where $V_a = \{w \in \{0,1\}^d | Q(w) = a\}$ for $a = 0, 1$.*

Let $H_Q = (V_0 \cup V_1, E)$ be a bipartite graph where $(u, v) \in V_0 \times V_1$ is an edge if $\mathsf{dist}(u, v) = 1$. Let $\mathcal{M}$ be all the possible matchings of H_Q. The size of the maximum matching of H_Q is

$$\mathsf{Match}(Q) = \max_{M \in \mathcal{M}} \Pr_v[\exists u, (u, v) \in M \text{ or } (v, u) \in M] = \max_{M \in \mathcal{M}} 2|M|/2^d,$$

where v is uniformly distributed in $V_0 \cup V_1$.

Definition 1.8.4 (Collection of Functions)**.** *For $s = s(n), m = m(n)$, a collection of functions $F : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^m$ takes an input (k, x) and outputs $F(k, x)$. Here k is a public index and x can be viewed as the input for the kth function in the collection. We also denote $F(k, x)$ as $F_k(x)$ where F_k is the kth function in the collection.*

Remark 1.8.5. *For simplicity, we usually consider n as an exponential of 2.*

In the following paragraph, an efficient adversary is defined to be a probabilistic

polynomial time Turing Machine. Also the term efficient means in probabilistic polynomial time.

Definition 1.8.6 (Approximate One-way Function for Collection of Functions)**.** *For $\delta = \delta(n) \in (0,1)$ and $\epsilon = \epsilon(n) \in (0,1)$, a collection of functions $F : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^m$ is an (ϵ, δ)-approximate one-way function if for every efficient adversary A which outputs a list of $\mathsf{poly}(n)$ candidates and for sufficiently large n's, we have that*

$$\Pr_{k,x,y=F_k(x)}[\exists z \in A(k,y), z' \in F_k^{-1}(y), \mathsf{dist}(z,z')/n \leq \delta] < \epsilon,$$

where k and x are independent and uniform. Specially, when $\delta = 0$, we say the collection F is ϵ-one-way.

Definition 1.8.7 (Goldreich's Random Local Function [Gol11])**.** *Given a predicate $Q : \{0,1\}^d \to \{0,1\}$ and an (n,m,d) hypergraph $G = ([n], S_0, \ldots, S_{m-1})$, the function $f_{G,Q} : \{0,1\}^n \to \{0,1\}^m$ is defined as follows: for input x, the ith output bit of $f_{G,Q}(x)$ is $f_{G,Q}(x)_i = Q(x_{S_i})$.*

For $m = m(n)$, the function collection $F_{Q,n,m} : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^m$ is defined via the mapping $(G,x) \to f_{G,Q}(x)$.

Lemma 1.8.8. *For every $d = O(\log n)$, every $m = \mathsf{poly}(n)$ and every predicate $Q : \{0,1\}^d \to \{0,1\}$, the random local function $F_{Q,n,m}$, following Definition 1.8.7, is in $\mathcal{AC}^0$.*

Proof. For every $i \in [m]$, we claim that the ith output bit of $F_{Q,n,m}(G,x)$ can be computed in $\mathcal{AC}^0$. The reason is as follows. We know that $F_{Q,n,m}(G,x)_i = Q(x_{S_i})$. So it is determined by d bits of x and S_i which corresponds to $d \log n$ bits of G. Thus

for $S_i = (j_0, j_1, \ldots, j_{d-1})$, $\forall l \in [d]$, the lth input for Q is

$$x_{j_l} = \bigvee_{k=0}^{n} (I_{j_l=k} \wedge x_k) = \bigwedge_{k=0}^{n} (I_{j_l \neq k} \vee x_k).$$

As $|j_l| = \log n$, $I_{j_l=k}$ and $I_{j_l \neq k}$ can be computed in $\mathcal{AC}^0$ by Lemma 1.2.11. So every input bit in x_{S_i} can be computed in $\mathcal{AC}^0$. As $d = O(\log n)$, we know that $F_{Q,n,m}(G, x)_i = Q(x_{S_i})$ can be computed in $\mathcal{AC}^0$. Thus $F_{Q,n,m}(G, x)$ can be computed in $\mathcal{AC}^0$.

□

Definition 1.8.9. *Two distribution ensembles $Y = \{Y_n\}$ and $Z = \{Z_n\}$ are ϵ-indistinguishable if for every efficient adversary A,*

$$|\Pr[A(1^n, Y_n) = 1] - \Pr[A(1^n, Z_n) = 1]| \leq \epsilon(n).$$

Here the subscript of a random variable indicates its length.

Definition 1.8.10 (PRG for a Collection of Functions)**.** *Let $m = m(n)$. A collection of functions $F : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^m$ is an ϵ-PRG, if $(K, F_K(U_n))$ is ϵ-indistinguishable from the uniform distribution. Here K is uniform over $\{0,1\}^s$, U_n is uniform over $\{0,1\}^n$.*

A collection of functions $F : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^m$ is ϵ-unpredictable generator (UG) if for every efficient adversary A and every sequence of indices $\{i_n\}_{n \in \mathbb{N}}$ where $i_n \in [m(n)]$, we have that

$$\Pr_{k \leftarrow U_s, x \leftarrow U_n}[A(k, F_k(x)_{[0,\ldots,i_n-1]}) = F_k(x)_{i_n}] \leq \epsilon(n)$$

for sufficiently large n's. Here F is ϵ-last-bit unpredictable generator (LUG) if $i_n = m(n) - 1$.

Remark 1.8.11. *Let $t = t(r)$. A function $G : \{0,1\}^r \to \{0,1\}^t$ is a classic ϵ-PRG, if $(K, F_K(U_n))$ is ϵ-indistinguishable from the uniform distribution. Here K is uniform over $\{0,1\}^r$, U_n is uniform over $\{0,1\}^n$.*

The definition of PRG for a collection of functions implies the classic definition of PRG. Following our definition, if there exists an explicit ϵ-PRG $F(\cdot,\cdot)$ for a collection of functions, we know $(U_s, F_{U_s}(U_n))$ is ϵ-indistinguishable from uniform distributions. Let $G : \{0,1\}^{r=s+n} \to \{0,1\}^{t=s+m}$ be such that $\forall k \in \{0,1\}^s, \forall x \in \{0,1\}^n, G(k \circ x) = k \circ F(k,x)$. We know that $G(U_r)$ is indistinguishable from uniform distributions. So G is a classic ϵ-PRG.

Definition 1.8.12. *An ϵ-LPRG is an ϵ-PRG whose output length is linear of its input length (including the index length, $m > (1+\delta)(n+s)$ for some constant δ).*

An ϵ-PPRG is an ϵ-PRG whose output length is a polynomial of its input length (including the index length, $m > (n+s)^{(1+\delta)}$ for some constant δ).

Lemma 1.8.13. *For every $c \in \mathbb{N}^+$, an ϵ-PRG $G : \{0,1\}^r \to \{0,1\}^t$ in $\mathcal{AC}^0$ can be transformed to an $(c\epsilon)$-PRG $G' : \{0,1\}^r \to \{0,1\}^{t(t/r)^c}$.*

Here $G'(\cdot) = G^{(c)}(\cdot)$, where $G^{(i+1)}(\cdot) = G^{(i)}(\cdot), \forall i \in \mathbb{N}^+$ and $G^{(1)}(\cdot) = G(\cdot)$.

If c is a constant, then G' is in $\mathcal{AC}^0$.

Proof. We use inductions. Assume the the output length for $G^{(i)}$ is $t^{(i)}$.

For the basic step, as G is an ϵ-PRG, $G^{(1)} = G$ is an ϵ-PRG.

For the induction step, assume for i, $G^{(i)}$ is an $(i\epsilon)$-PRG. Suppose there exists an efficient adversary A such that

$$|\Pr[A(G^{(i+1)}(U_r)) = 1] - \Pr[A(U_{t^{(i+1)}}) = 1]| > (i+1)\epsilon.$$

We know that

$$|\Pr[A(G^{(i+1)}(U_r)) = 1] - \Pr[A(U_{t^{(i+1)}}) = 1]|$$

$$\leq |\Pr[A(G^{(i+1)}(U_r)) = 1] - \Pr[A(G(U_{t^{(i)}})) = 1]| + |\Pr[A(G(U_{t^{(i)}})) = 1] - \Pr[A(U_{t^{(i+1)}}) = 1]| \tag{1.23}$$

As $|\Pr[A(G(U_{t^{(i)}})) = 1] - \Pr[A(U_{t^{(i+1)}}) = 1]| \leq \epsilon$,

$$|\Pr[A(G^{(i+1)}(U_r)) = 1] - \Pr[A(G(U_{t^{(i)}})) = 1]| > i\epsilon$$

contradicting the the induction assumption. So $G^{(i+1)}$ is an $(i+1)\epsilon$-PRG. □

Theorem 1.8.14. *For any d-ary predicate Q, if the random local function $F_{Q,n,m}$ is δ-one-way for some constant $\delta \in (0,1)$, then we have the following results.*

1. *For some constant $c = c(d) > 1$, if $m > cn$, then there exists a ϵ-LPRG in $\mathcal{AC}^0$ with ϵ being negligible.*
2. *For any constant $c > 1$, if $m > n^c$, then there exists a ϵ-PPRG in $\mathcal{AC}^0$ with ϵ being negligible.*

Before we prove Theorem 1.8.14, we first use it to obtain our main theorem in this subsection.

Theorem 1.8.15. *For any d-ary predicate Q, if the random local function $F_{Q,n,m}$ is δ-one-way for some constant $\delta \in (0,1)$, then we have the following results.*

1. *For some constant $c > 1$, if $m > cn$, then for any constant $a > 1$, there exists a ϵ-LPRG $G : \{0,1\}^r \to \{0,1\}^t$ in $\mathcal{AC}^0$, where $t \geq ar$ and ϵ is negligible.*
2. *For any constant $c > 1$, if $m > n^c$, then for any constant $a > 1$ there exists a ϵ-PPRG $G : \{0,1\}^r \to \{0,1\}^t$ in $\mathcal{AC}^0$, where $t \geq r^a$ and ϵ is negligible.*

Proof. For the first assertion, let the LPRG in Theorem 1.8.14 be $G_0 : \{0,1\}^{r_0} \to \{0,1\}^{t_0}$ with $t_0 > c_0 r_0$ for some constant $c_0 > 1$. We apply the construction in Lemma 1.8.13 to obtain $G^{(c_1)}$ such that $c_0^{c_1} \geq a$. So c_1 is a constant. By Lemma 1.8.13 we know that $G^{(c_1)}$ is a $c_1\epsilon$-PRG in $\mathcal{AC}^0$. This proves the first assertion.

By the same reason, the second assertion also holds. □

Construction 1.8.16. *Let $F_{Q,n,m} : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^m$ be the random local function following Definition 1.8.7. We construct $F' : \{0,1\}^s \times \{0,1\}^{n'} \to \{0,1\}^{m'}$ where $n' = tn, m' = tm, t = n$.*

1. *Draw G uniformly from $\{0,1\}^s$.*
2. *Draw $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$ independently uniformly from $\{0,1\}^n$. Let $x = (x^{(1)}, x^{(2)}, \ldots, x^{(t)})$.*
3. *Output $F'(G,x) = \bigcirc_{i=1}^{t} G(x^{(i)})$.*

Lemma 1.8.17. *In Construction 1.8.16, for every constant $d \in \mathbb{N}^+$, every predicate $Q : \{0,1\}^d \to \{0,1\}$, every $m = \mathsf{poly}(n)$ and every $\epsilon = 1/\mathsf{poly}(n)$, if $F_{Q,n,m}$ is $(\frac{1}{2} + \epsilon)$-last-bit unpredictable then F' is $(\frac{1}{2} + \epsilon(1 + 1/n))$-unpredictable.*

Proof. Suppose there exists a next-bit predictor P and a sequence of indices $\{i_n\}$ such that

$$\Pr_{x \leftarrow U_n, G \leftarrow U_s, y = F'(G,x)}[P(G, y_{0,\ldots,i_n-1}) = y_{i_n}] \geq \frac{1}{2} + \epsilon(n)(1 + 1/n)$$

for sufficiently large n's.

Now we construct a last-bit predicator P' which can predicate the last bit of $F_{Q,n,m}$ with success probability $1/2 + \epsilon$.

By Remark 3.2 of [App13], P' can find an index $j \in [m']$ by running a randomized algorithm M in polynomial time, such that, with probability $1 - 2^{-\Theta(n)}$ over the random bits used in M,

$$\Pr_{G,x,y=F'(G,x)}[P(G, y_{[0,\ldots,j-1]}) = y_j] > \frac{1}{2} + \epsilon(n) + \frac{\epsilon(n)}{2n}.$$

Recall that in Remark 3.2 of [App13], M' tries every index and pick the best one.

According to Construction 1.8.16, assume $j = an + b$ for some $a, b \in \mathbb{N}, b < n$.

Given $(G, y_{[0,\ldots,m-2]})$, P' generates $x^{(1)}, x^{(2)}, \ldots, x^{(a-1)}$ independently uniformly over $\{0,1\}^n$. Also P' constructs a hypergraph G' by swapping S_b and S_{m-1} of G. Next, P' computes $y' = \bigcirc_{i=1}^{a} G'(x^{(i)}) \circ y_{[0,\ldots,b-2]}$. Finally P' outputs $P(G', y')$. As G is uniform, G' is also uniform. Also as $x^{(1)}, \ldots, x^{(a)}$ are uniform, (G', y') has the same distribution as $(G, y_{[0,\ldots,j-1]})$. So

$$\Pr[P'(G', y') = y_{m-1}] \geq \frac{1}{2} + \epsilon(n) + \frac{\epsilon(n)}{2n} - 2^{\Theta(n)} > \frac{1}{2} + \epsilon(n).$$

This contradicts that $F_{Q,n,m}$ is $(\frac{1}{2} + \epsilon)$-last-bit unpredictable. □

Theorem 1.8.18 ([App13], Section 5)**.** *For every constant $d \in \mathbb{N}$, predicate $Q : \{0,1\}^d \to \{0,1\}$, and constant $\epsilon \in (0, \mathsf{Match}(Q)/2)$, there exists a constant $c > 0$ such that for every polynomial $m > cn$ the following holds. If the collection $F_{Q,n,m}$ is $\epsilon/5$-one-way then it is a $(1 - \mathsf{Match}(Q)/2 + \delta)$-last-bit UG where $\delta = \epsilon(1 - o(1))$. Thus it is also a $(1 - \mathsf{Match}(Q)/2 + \epsilon)$-UG.*

Remark 1.8.19. *Our definition of random local function has only one difference with the definition of [App13]. That is, for each hyperedge we do not require the incoming vertices to be distinct. This difference does not affect the correctness of Theorem 1.8.18.*

Construction 1.8.20 (Modified from [App13] Construction 6.8)**.** *Let* $F : \{0,1\}^{s(n)} \times \{0,1\}^n \to \{0,1\}^{m(n)}$ *be a UG and* $\mathsf{Ext} : \{0,1\}^{n_1} \times \{0,1\}^{d_1} \to \{0,1\}^{m_1}$ *be a strong* $(k = \alpha n_1, \epsilon_1)$*-extractor following Theorem 1.6.16 where* $n_1 = n$*,* α *is some constant,* $\epsilon = 1/2^{\Theta(\log^a n)}$ *for some large enough constant* $a \in \mathbb{N}^+$*,* $d_1 = (\log a)^{\Theta(a)}$*,* $m = 0.9k$*.*

We construct the following UG $H : \{0,1\}^{sn} \times \{0,1\}^{n^2+d_1 n} \to \{0,1\}^{mn}$*.*

1. *Index: Generate* $G_0, G_1, \ldots, G_{n-1}$ *independently uniformly over* $\{0,1\}^s$*. Generate extractor seeds* $u_0, u_1, \ldots, u_{m-1}$ *independently uniformly over* $\{0,1\}^{d_1}$*. Denote* $G = (G_0, G_1, \ldots, G_{n-1})$ *and* $u = (u_0, u_1, \ldots, u_{m-1})$*.*

2. *Input: Generate* $x^{(0)}, x^{(1)}, \ldots, x^{(n-1)}$ *independently uniformly over* $\{0,1\}^n$*. Denote* $x = (x^{(0)}, x^{(1)}, \ldots, x^{(n-1)})$*.*

3. *Output: Compute the* $n \times m$ *matrix* Y *whose ith row is* $G_i(x^{(i)})$*. Let* Y_i *denote the ith column of* Y*. Output* $H(G, u, x) = \mathsf{Ext}(Y_0, u_0) \circ \mathsf{Ext}(Y_1, u_1) \circ \cdots \circ \mathsf{Ext}(Y_{m-1}, u_{m-1})$*.*

Remark 1.8.21. *There are 2 differences between our construction and Construction 6.8 of [App13]. First, we use our AC0-extractor to do extraction. As our extractor is strong, its seed can also be regarded as part of the public key (index). Second, our construction is for any* m*, while their construction only considers* m *as a linear function of* n*.*

Lemma 1.8.22. *For any constant* $\epsilon \in [0, 1/2)$*, if* F *is* $(\frac{1}{2} + \epsilon)$*-unpredictable, then the mapping* H *is a PRG with negligible error.*

Proof Sketch. The proof is almost the same as that of Lemma 6.9 of [App13].

By the same argument of Lemma 6.9 of [App13], we know that for every sequence of efficiently computable index family $\{i_n\}$ and every efficient adversary A, there exists a random variable $W \in \{0,1\}^n$ jointly distributed with G and Y such that

- the min-entropy of W, given any fixed G and the first i_n columns of Y, is at least $n(1-2\epsilon-o(1))$.
- A cannot distinguish between $(G, Y_{[0,\dots,i_n]})$ and $(G, [Y_{[0,\dots,i_n-1]}W])$ with more than negligible advantage even when A is given an oracle which samples the distribution (G,Y,W). Here $[Y_{[0,\dots,i_n-1]}W]$ is a matrix such that the first i_n columns are $Y_{[0,\dots,i_n-1]}$ and the last column is W.

By the definition of strong extractors, for every family $\{i_n\}$, the distribution

$$(G, u, Y_{[0,\dots,i_n-1]}, \mathsf{Ext}(Y_{i_n}, u_{i_n}))$$

is indistinguishable from $(G, u, Y_{[1,\dots,i_n-1]}, U_{m_1})$. Otherwise, suppose there is an adversary B that can distinguish the two distributions. We construct another adversary A as the follows. First A generates a uniform u as seeds for the extractors and invokes B on $(G, u, y, \mathsf{Ext}(v,u))$ where G is generated from uniform, y is drawn from $Y_{[0,\dots,i_n-1]}$. If v is drawn from Y_{i_n} then B gets a sample from $(G, u, Y_{[0,\dots,i_n-1]}, \mathsf{Ext}(Y_{i_n}, u))$. If v is drawn from W, then B gets a sample from $(G, u, Y_{[0,\dots,i_n-1]}, \mathsf{Ext}(W,u))$ which is ϵ_0-close to $(G, u, Y_{[0,\dots,i_n-1]}, U_{m_1})$ by the definition of strong extractors, where ϵ_0 is negligible according to our settings in Construction 1.8.20 and U_{m_1} is the uniform distribution of length m_1. So A can distinguish $(G, Y_{[0,\dots,i_n]})$ and $(G, [Y_{[0,\dots,i_n-1]}W])$, having the same distinguishing advantage as B does (up to a negligible loss). This is a contradiction.

As a result, for every family $\{i_n\}$, the distributions

$$(G, u, H(G, u, x)_{[0,\ldots,i_n]}) \text{ and } (G, u, H(G, u, x)_{[0,i_{n-1}]} \circ U_{m_1})$$

are indistinguishable. So H is a $(1/2 + \mathsf{neg}(n))$-UG. By Fact 6.1 (Yao's theorem) of [App13], H is a PRG. □

Proof of Theorem 1.8.14. We combine Construction 1.8.16 and Construction 1.8.20 together by using the UG of Construction 1.8.16 in Construction 1.8.20. By Theorem 1.8.18 and Lemma 1.8.22, we know that our construction gives a PRG (with negligible error). Assume the PRG is $H : \{0,1\}^{s_H} \times \{0,1\}^{n_H} \to \{0,1\}^{m_H}$.

Next we mainly focus on the stretch. The output length of H is $m_H = \Theta(ntm)$, the input length (including the index length) is $s_H + n_H = sn + d_1 n + n^2 t$. Here we know that $s = m \log n$, $t = n$.

Assume $m > cn$ for some constant $c > 1$. We know that $\frac{m_H}{s_H + n_H} = c' > 1$, for some constant c'.

For the polynomial stretch case, assume $m > n^c$ for some constant $c > 1$. We know that $m_H \geq (s_H + n_H)^{c'}$ for some constant $c' > 1$.

For both cases, the construction is in $\mathcal{AC}^0$. The reason is as follows. By Lemma 1.8.8, the random local function is in $\mathcal{AC}^0$. In Construction 1.8.16 and Construction 1.8.20, we compute $O(nt)$ random local functions (some of them share the same index) in parallel. Also our extractor is in $\mathcal{AC}^0$. So the overall construction is in $\mathcal{AC}^0$.

This proves the theorem.

□

1.8.2 PRG in AC0 for Space Bounded Computation

In this subsection, we give an $\mathcal{AC}^0$ version of the PRG in [NZ96].

Theorem 1.8.23. *For every constant $c \in \mathbb{N}$ and every $m = m(s) = \mathsf{poly}(s)$, there is an explicit PRG $g : \{0,1\}^{r=O(s)} \to \{0,1\}^m$ in $\mathcal{AC}^0$, such that for any randomized algorithm A using space s,*

$$|\Pr[A(g(U_r)) = 1] - \Pr[A(U_m) = 1]| = \epsilon \leq 2^{-\Theta(\log^c s)},$$

where U_r is the uniform distribution of length r, U_m is the uniform distribution of length m.

Proof Sketch. We modify the construction of [NZ96] by replacing their extractor with the extractor from Theorem 1.6.16 for some constant entropy rate and with error parameter $\epsilon' = 2^{-\Theta(\log^c s)}$. In the PRG construction of [NZ96], it only requires an extractor for constant entropy rate. As our extractor meets their requirement, the proof in [NZ96] still holds under this modification.

For the security parameter ϵ, according to [NZ96], $\epsilon = \mathsf{poly}(s)(\epsilon' + 2^{-s})$. As a result, $\epsilon = 2^{-\Theta(\log^c s)}$.

□

Chapter 2

Secret Sharing in $\mathcal{AC}^0$

2.1 Introduction

The motivation for this chapter comes from two different sources. The first is the general theme of improving performance at the price of allowing some small probability of error or failure. This is evident throughout computer science. For example, randomized algorithms tend to be much more efficient than their deterministic counterparts. In cryptography and coding theory, randomization with small failure probability can often be used to amplify security or improve efficiency. This is arguably a good tradeoff in practice.

The second source of motivation is the goal of minimizing the computational complexity of cryptographic primitives and related combinatorial objects. For example, a line of work on the parallel complexity of cryptography [Gol11; CM01; MST06; AIK06; AIK08] successfully constructed one-way functions and other cryptographic primitives in the complexity class $\mathcal{NC}^0$ based on different kinds of assumptions, including very standard cryptographic assumptions. Works along this line have found several unexpected applications, most recently in the context of general-purpose

obfuscation [Lin16]. The study of low-complexity cryptography is also motivated by the goal of obtaining stronger *negative results*. For instance, low-complexity pseudo-random functions imply stronger hardness results for learning [NR99] and stronger natural proof barriers [MV15], and low-complexity decryption [BL16] implies a barrier for function secret sharing [BGI15].

In this chapter, we address the question of minimizing the complexity of secret sharing schemes and error correcting codes by introducing additional randomization and allowing for a small failure probability. We focus on the complexity class $\mathcal{AC}^0$, which is the lowest class for which a secret can be reconstructed or a message be decoded with negligible error probability. We show that the randomization approach can be used towards obtaining much better parameters than previous constructions. In some cases, our parameters are close to optimal and would be impossible to achieve without randomization.

We now give a more detailed account of our results, starting with some relevant background.

2.1.1 (Robust) secret sharing in $\mathcal{AC}^0$

A secret sharing scheme allows a dealer to randomly split a secret between n parties so that qualified subsets of parties can reconstruct the secret from their shares while unqualified subsets learn nothing about the secret. We consider here a variant of threshold secret sharing (also known as a "ramp scheme"), where any k parties can learn nothing about the secret, whereas all n parties together can recover the secret from their shares. We also consider a robust variant where the secret should be correctly reconstructed even if at most d shares are corrupted by an adversary, possibly

in an adaptive fashion. We formalize this below.

Definition 2.1.1 (secret sharing)**.** *An (n,k) secret sharing scheme with message alphabet Σ_0, message length m, and share alphabet Σ is a pair of functions* $(\mathsf{Share}, \mathsf{Rec})$*, where* $\mathsf{Share} : \Sigma_0^m \to \Sigma^n$ *is probabilistic and* $\mathsf{Rec} : \Sigma^n \to \Sigma_0^m$ *is deterministic, which satisfy the following properties.*

- *Privacy: For a privacy threshold k, the adversary can choose a sequence $W = (w_1, \ldots, w_k) \in [n]^k$ of share indices to observe, either adaptively (where each w_i depends on previously observed shares $\mathsf{Share}(x)_{w_1}, \ldots, \mathsf{Share}(x)_{w_{i-1}}$) or non-adaptively (where W is picked in one shot). We say that the scheme is ϵ-private if for every such strategy, there is a share distribution $\mathcal{D}$ over Σ^k such that for every secret message $x \in \Sigma_0^m$, $\mathsf{Share}(x)_W$ is ϵ-close (in statistical distance) to $\mathcal{D}$. We refer to ϵ as the* privacy error *and say that the scheme has* perfect privacy *if $\epsilon = 0$.*

- *Reconstruction: We say that the scheme has reconstruction error η if for every $x \in \Sigma_0^m$,*

$$\Pr[\mathsf{Rec}(\mathsf{Share}(x)) = x] \geq 1 - \eta.$$

  We say the scheme has perfect reconstruction *if $\eta = 0$.*

We are also interested in robust secret sharing*, where an adversary is allowed to modify at most d shares.*

- *Robustness: For any secret $x \in \Sigma_0^m$, let $Y = \mathsf{Share}(x)$. Consider an arbitrary adversary who (adaptively or non-adaptively) observes d shares and can then arbitrarily change these d shares, transforming Y to Y'. The scheme is d-*robust

if for every such adversary,

$$\Pr[\mathsf{Rec}(Y') = x] \geq 1 - \eta.$$

If the share alphabet and the message alphabet are both Σ, then we simply say the alphabet of the scheme is Σ. By saying that a secret sharing scheme is in $\mathcal{AC}^0$, we mean that both the sharing function and the reconstruction function can be computed by (uniform) $\mathcal{AC}^0$ circuits.

A recent work of Bogdanov et al. [Bog+16] considers the complexity of sharing and reconstructing secrets. The question is motivated by the observation that almost all known secret sharing schemes, including the well known Shamir's scheme [Sha79], require the computation of linear functions over finite fields, and thus cannot be implemented in the class $\mathcal{AC}^0$ (i.e., constant depth circuits). Thus a natural question is whether there exist secret sharing schemes in $\mathcal{AC}^0$ with good parameters. In the case of threshold secret sharing, Bogdanov et. al [Bog+16] showed a relation between the approximate degree[1] of a function and the privacy threshold of a secret sharing scheme. Using this and known approximate degree lower bounds, they obtained several secret sharing schemes with sharing and reconstruction functions computable in $\mathcal{AC}^0$. However, to achieve a large privacy threshold (e.g., $k = \Omega(n)$) their construction needs to use a large alphabet (e.g., size $2^{\mathsf{poly}(n)}$). In the case of binary alphabet, they can only achieve privacy threshold $\Omega(\sqrt{n})$ with perfect reconstruction and privacy threshold $\Omega((n/\log n)^{2/3})$ with constant reconstruction error $\eta < 1/2$. This limit is inherent without improving the best known approximate degree of an $\mathcal{AC}^0$ function [BT17]. Furthermore, their schemes only share one bit, and a naive

[1] The approximate degree of a Boolean function is the lowest degree of a real polynomial that can approximate the function within, say, an additive difference of $1/3$ on every input.

approach of sharing more bits by repeating the scheme multiple times will lead to a bad information rate. This leaves open the question of improving these parameters. Ideally, we would like to share many bits (e.g., $\Omega(n)$), obtain a large privacy threshold (e.g., $\Omega(n)$), and achieve perfect reconstruction and small alphabet size at the same time.

In order to improve the $\mathcal{AC}^0$ secret sharing schemes from [Bog+16], we relax their perfect privacy requirement and settle for the notion of ϵ-privacy from Definition 2.1.1. (This relaxation was recently considered in [BW17], see discussion below.) Note that this relaxation is necessary to improve the privacy threshold of $\mathcal{AC}^0$ secret sharing schemes, unless one can obtain better approximate degree lower bounds of an explicit $\mathcal{AC}^0$ function (as [Bog+16] showed that an explicit $\mathcal{AC}^0$ secret sharing scheme with privacy threshold k and perfect privacy also implies an explicit function in $\mathcal{AC}^0$ with approximate degree at least k). Like most schemes in [Bog+16], we only require that the secret can be reconstructed by all n parties. On the other hand, we always require perfect reconstruction. We show that under this slight relaxation, we can obtain much better secret sharing schemes in $\mathcal{AC}^0$. For an adaptive adversary, we can achieve both a constant information rate and a large privacy threshold ($k = \Omega(n)$) over a binary alphabet. In addition, our privacy error is exponentially small. Specifically, we have the following theorem.

Theorem 2.1.2 (adaptive adversary)**.** *For every $n \in \mathbb{N}$ and constant $\gamma \in (0, 1/4)$, there exists an explicit $(n, \Omega(n))$ secret sharing scheme in $\mathcal{AC}^0$ with alphabet $\{0,1\}$, secret length $m = \Omega(n)$, adaptive privacy error $2^{-\Omega(n^{\frac{1}{4}-\gamma})}$ and perfect reconstruction.*

Note that again, by using randomization and allowing for a small privacy error, we

can significantly improve both the privacy threshold and the information rate, while also making the scheme much more efficient by using a smaller alphabet.

Remark 2.1.3. *We note that a recent paper by Bun and Thaler [BT17] gave improved lower bounds for the approximate degree of $\mathcal{AC}^0$ functions. Specifically, for any constant $\alpha > 0$ they showed an explicit $\mathcal{AC}^0$ function with approximate degree at least $n^{1-\alpha}$, and by the relation established in [Bog+16] this also gives a secret sharing scheme in $\mathcal{AC}^0$ with privacy threshold $n^{1-\alpha}$. However, our results are stronger in the sense that we can achieve threshold $\Omega(n)$, and furthermore we can achieve perfect reconstruction while the secret sharing scheme in [BT17] only has constant reconstruction error.*

Remark 2.1.4. *Our construction of $\mathcal{AC}^0$ secret sharing schemes is actually a general transformation and can take any such scheme in [Bog+16] or [BT17] as the starting point. The error $2^{-\Omega(n^{\frac{1}{4}-\gamma})}$ in Theorem 2.1.2 comes from our use of the one-in-a-box function [MP88], which has approximate degree $n^{1/3}$. We can also use the new $\mathcal{AC}^0$ function of [BT17] with approximate degree $n^{1-\alpha}$, which will give us an error of $2^{-\Omega(n^{\frac{1}{2}-\gamma})}$ but the reconstruction error will become a constant. We note that the privacy error of our construction is also close to optimal, without further improvement on the lower bounds of approximate degree of $\mathcal{AC}^0$ functions. This is because a privacy error of 2^{-s} will imply an $\mathcal{AC}^0$ function of approximate degree $\Omega(s/\log n)$. Thus if one can achieve a sufficiently small privacy error (e.g., $2^{-\Omega(n)}$), then this will give an improved approximate degree lower bound for an $\mathcal{AC}^0$ function.*

A very recent paper by Bogdanov and Williamson [BW17] considered a similar relaxation as ours. Specifically, they showed how to construct two distributions over n bits that are (k, ϵ)-wise indistinguishable, but can be distinguished with advantage $1 -$

η by some $\mathcal{AC}^0$ function. Here (k,ϵ)-wise indistinguishable means that if looking at any subset of k bits, the two distributions have statistical distance at most ϵ. Translating into the secret sharing model, this roughly implies an $\mathcal{AC}^0$ secret sharing scheme with binary alphabet, privacy threshold k, privacy error ϵ and reconstruction error η. Bogdanov and Williamson [BW17] obtained several results in this case. Specifically, they showed a pair of such distributions for any $k \leq n/2$ with $\epsilon = 2^{-\Omega(n/k)}$, that can be distinguished with $\eta = \Omega(1)$ by the OR function; or for any k with $\epsilon = 2^{-\Omega((n/k)^{1-1/d})}$, that can be distinguished with $\eta = 0$ by a depth-d AND-OR tree.

We note the following important differences between our results and the corresponding results by Bogdanov and Williamson [BW17]: first, the results in [BW17], in the language of secret sharing, only consider a 1-bit secret, while our results can share $\Omega(n)$ bits with the same share size. Thus our information rate is much larger than theirs. Second, we can achieve a privacy threshold of $k = \Omega(n)$ while simultaneously achieving an exponentially small privacy error of $\epsilon = 2^{-n^{\Omega(1)}}$ and perfect reconstruction ($\eta = 0$). In contrast, the results in [BW17], when going into the range of $k = \Omega(n)$, only have constant privacy error. In short, our results are better than the results in [BW17], in the sense that we can simultaneously achieve asymptotically optimal information rate and privacy threshold, exponentially small privacy error and perfect reconstruction. As a direct corollary, we have the following result, which is incomparable to the results in [BW17].

Corollary 2.1.5. *There exists a constant $\alpha > 0$ such that for every n and $k \leq \alpha n$, there exists a pair of $(k, 2^{-n^{\Omega(1)}})$-wise indistinguishable distributions X, Y over $\{0,1\}^n$ and an $\mathcal{AC}^0$ function D such that* $\Pr[D(X)] - \Pr[D(Y)] = 1$.

Next, we extend our $\mathcal{AC}^0$ secret sharing schemes to the robust case, where the adversary can tamper with several parties' shares. Our goal is to simultaneously achieve a large privacy threshold, a large tolerance to errors, a large information rate and a small alphabet size. We can achieve a constant information rate with privacy threshold and error tolerance both $\Omega(n)$, with constant size alphabet, exponentially small privacy error and polynomially small reconstruction error. However, here we can only handle a non-adaptive adversary. Specifically, we have the following theorem.

Theorem 2.1.6 (non-adaptive adversary)**.** *For every $n \in \mathbb{N}$, every $\eta = \frac{1}{\mathsf{poly}(n)}$, there exists an explicit $(n, \Omega(n))$ robust secret sharing scheme in $\mathcal{AC}^0$ with share alphabet $\{0,1\}^{O(1)}$, message alphabet $\{0,1\}$, message length $m = \Omega(n)$, non-adaptive privacy error $2^{-n^{\Omega(1)}}$, non-adaptive robustness $\Omega(n)$ and reconstruction error η.*

2.1.2 Error correcting codes for additive channels in $\mathcal{AC}^0$

Robust secret sharing schemes are natural generalizations of error correcting codes. Thus our robust secret sharing schemes in $\mathcal{AC}^0$ also give error correcting codes with randomized $\mathcal{AC}^0$ encoding and deterministic $\mathcal{AC}^0$ decoding. The model of our error correcting codes is the same as that considered by Guruswami and Smith [GS16]: stochastic error correcting codes for additive channels. Here, the code has a randomized encoding function and a deterministic decoding function, while the channel can add an arbitrary error vector $e \in \{0,1\}^n$ of Hamming weight at most ρn to the transmitted codeword of length n. As in [GS16], the error may depend on the message but crucially does not depend on the randomness used by the encoder. Formally, we have the following definition.

Definition 2.1.7. *For any $n, m \in \mathbb{N}$, any $\rho, \epsilon > 0$, an (n, m, ρ) stochastic binary error correcting code $(\mathsf{Enc}, \mathsf{Dec})$ with randomized encoding function $\mathsf{Enc} : \{0,1\}^m \to \{0,1\}^n$, deterministic decoding function $\mathsf{Dec} : \{0,1\}^n \to \{0,1\}^m$ and decoding error ϵ, is such that for every $x \in \{0,1\}^m$, every $e = (e_1, \ldots, e_m) \in \{0,1\}^m$ with hamming weight at most ρn,*

$$\Pr[\mathsf{Dec}(\mathsf{Enc}(x) + e) = x] \geq 1 - \epsilon.$$

An (n, m, ρ) stochastic error correcting code $(\mathsf{Enc}, \mathsf{Dec})$ can be computed by $\mathcal{AC}^0$ circuits if both Enc and Dec can be computed by $\mathcal{AC}^0$ circuits.

Guruswami and Smith [GS16] constructed such codes that approach the Shannon capacity $1 - H(\rho)$. Their encoder and decoder run in polynomial time and have exponentially small decoding error. Here, we aim at constructing such codes with $\mathcal{AC}^0$ encoder and decoder. In a different setting, Goldwasser et. al [Gol+07] gave a construction of *locally decodable* codes that can tolerate a constant fraction of errors and have $\mathcal{AC}^0$ decoding. Their code has deterministic encoding but randomized decoding. By repeating the local decoder for each bit for $O(\log n)$ times and taking majority, one can decode each bit in $\mathcal{AC}^0$ with error probability $1/\mathsf{poly}(n)$ and thus by a union bound the original message can also be decoded with error probability $1/\mathsf{poly}(n)$. However we note that the encoding function of [Gol+07] is not in $\mathcal{AC}^0$, and moreover their message rate is only polynomially small. In contrast, our code has constant message rate and can tolerate a constant fraction of errors (albeit in a weaker model) when the decoding error is $1/\mathsf{poly}(n)$ or even $2^{-\mathsf{poly}\log(n)}$. The rate and tolerance are asymptotically optimal. We can achieve even smaller error $(2^{-\Omega(r/\log n)})$ with message rate $1/r$. Furthermore both our encoding and decoding

are in $\mathcal{AC}^0$. Specifically, we have the following theorems.

Theorem 2.1.8 (error-correcting codes)**.** *For any $n \in \mathbb{N}$ and $\epsilon = 2^{-\mathsf{poly}\log(n)}$, there exists an $(n, \Omega(n), \Omega(1))$ stochastic binary error correcting code with decoding error ϵ, which can be computed by $\mathcal{AC}^0$ circuits.*

Theorem 2.1.9 (error-correcting codes with smaller decoding error)**.** *For any $n, r \in \mathbb{N}$, there exists an $(n, m = \Omega(n/r), \Omega(1))$ stochastic binary error correcting code with decoding error $2^{-\Omega(r/\log n)}$, which can be computed by $\mathcal{AC}^0$ circuits.*

Note that Theorem 2.1.9 is interesting mainly in the case where r is at least $\mathsf{poly}\log n$.

Remark 2.1.10. *We note that, without randomization, it is well known that deterministic $\mathcal{AC}^0$ circuits cannot compute asymptotically good codes [LV11]. Thus the randomization in our $\mathcal{AC}^0$ encoding is necessary here. For deterministic $\mathcal{AC}^0$ decoding, only very weak lower bounds are known. In particular, Lee and Viola [LV15] showed that any depth-c $\mathcal{AC}^0$ circuit with parity gates cannot decode beyond error $(1/2 - 1/O(\log n)^{c+2})d$, where d is the distance of the code. While the repetition code can be decoded in $\mathcal{AC}^0$ with a near-optimal fraction of errors by using approximate majority, obtaining a similar positive result for codes with a significantly better rate is open.*

2.1.3 Secure broadcasting with an external adversary

We apply our ideas and technical approach to the following flavor of secure broadcasting in the presence of an adversary. The problem can be viewed as a generalization of a one-time pad encryption. In a one-time pad encryption, two parties share a secret key

which can be used to transmit messages with information-theoretic security. Suppose that each party wants to transmit an m-bit string to the other party. If an external adversary can see the entire communication, then it is well known that to keep both messages secret, the parties must share a secret key of length at least $2m$. This can be generalized to the case of n parties, where we assume that they have access to a public broadcast channel, and each party wants to securely communicate an m-bit string to all other parties. This problem can be useful, for example, when n collaborating parties want to compute a function of their secret inputs without revealing the inputs to an external adversary. Again, if the adversary can see the entire communication, then the parties need to share a secret key of length at least nm.

Now, what if we relax the problem by restricting the adversary's power? Suppose that instead of seeing the entire communication, the adversary can only see some fraction of the communicated messages. Can we get more efficient solutions? We formally define this model below, requiring not only the secrecy of the inputs but also correctness of the outputs in the presence of tampering with a bounded fraction of messages.

Definition 2.1.11. *Let $n, m \in \mathbb{N}$ and $\alpha, \epsilon > 0$. An $(n, m, \alpha, \epsilon, \eta)$-secure broadcasting protocol is an n-party protocol with the following properties. Initially, each party i has a local input $x_i \in \{0,1\}^m$ and the parties share a secret key. The parties can then communicate over a public broadcast channel. At the end of the communication, each party computes a local output. We require the protocol to satisfy the following security properties.*

- *(Privacy) For any adversarial observation W which observes at most $1 - \alpha$ fraction of the messages, there is a distribution $\mathcal{D}$, such that for any inputs*

$x = (x_1, \ldots, x_n) \in (\{0,1\}^m)^n$ *leading to a sequence of messages* Y*, the distribution* Y_W *of observed messages is* ϵ*-close to* $\mathcal{D}$*.*

- *(Robustness) For any adversary that corrupts at most* $1 - \alpha$ *fraction of the messages, and any* n*-tuple of inputs* $x = (x_1, \ldots, x_n) \in (\{0,1\}^m)^n$*, all* n *parties can reconstruct* x *correctly with probability at least* $1 - \eta$ *after the communication.*

The naive solution of applying one-time pad still requires a shared secret key of length at least nm, since otherwise even if the adversary only sees part of the communication, he may learn some information about the inputs. However, by using randomization and allowing for a small error, we can achieve much better performance. Specifically, we have the following theorem.

Theorem 2.1.12 (secure broadcasting)**.** *For any* $n, m, r \in \mathbb{N}$ *with* $r \leq m$*, there exists an explicit* $(n, m, \alpha = \Omega(1), n2^{-\Omega(r)}, n2^{-\Omega(r)} + nm2^{-\Omega(m/r)})$ *secure broadcasting protocol with communication complexity* $O(nm)$ *and shared secret key of length* $O(r \log(nr))$*.*

2.1.4 Overview of the techniques

Secret sharing. Here we give an overview of the techniques used in our constructions of $\mathcal{AC}^0$ secret sharing schemes and error correcting codes. Our constructions combine several ingredients in pseudorandomness and combinatorics in an innovative way, so before describing our constructions, we will first describe the important ingredients used.

The secret sharing scheme in [Bog+16]. As mentioned before, Bogdanov et. al [Bog+16] were the first to consider secret sharing schemes in $\mathcal{AC}^0$. Our constructions will use one of their schemes as the starting point. Specifically, since we aim at perfect reconstruction, we will use the secret sharing scheme in [MP88] based on the so called "one-in-a-box function" or Minsky-Papert CNF function. This scheme can share one bit among n parties, with binary alphabet, privacy threshold $\Omega(n^{1/3})$ and perfect reconstruction.

Random permutation. Another important ingredient, as mentioned before, is random permutation. Applying a random permutation, in many cases, reduces worst case errors to random errors, and the latter is much more convenient to handle. This property has been exploited for improving the efficiency of error correcting codes in several previous work, such as the error correcting codes by Smith [Smi07], Guruswami and Smith [GS16], and Hemenway et al. [Hem+11]. We note that a random permutation from $[n]$ to $[n]$ can be computed in $\mathcal{AC}^0$ [MV91; Hag91; Vio12].

K-wise independent generators. The third ingredient of our construction is the notion of k-wise independent pseudorandom generators. This is a function that stretches some r uniform random bits to n bits such that any subset of k bits is uniform. Such generators are well studied, while for our constructions we need such generators which can be computed by $\mathcal{AC}^0$ circuits. This requirement is met by using k-wise independent generators based on unique neighbor expander graphs, such as those constructed by Guruswami et. al [GUV09] which use seed length $r = k\text{poly}\log(n)$.

Secret sharing schemes based on error correcting codes. Using asymptotically good linear error correcting codes, one can construct secret sharing schemes that simultaneously achieve constant information rate and privacy threshold $\Omega(n)$ (e.g., [Che+07]). However, certainly in general these schemes are not in $\mathcal{AC}^0$ since they need to compute linear functions such as parity. For our constructions, we will use these schemes with a small block length (e.g., $O(\log n)$ or $\mathsf{poly}\log(n)$) such that parity with such input length can be computed by constant depth circuits. For robust secret sharing, we will also be using robust secret sharing schemes based on codes, with constant information rate, privacy threshold and tolerance $\Omega(n)$ (e.g., [Che16]), with a small block length.

The constructions. We can now give an informal description of our constructions. As mentioned before, our construction is a general transformation and can take any scheme in [Bog+16] or [BT17] as the starting point. A specific scheme of interest is the one in [Bog+16] based on the one-in-a-box function, which has perfect reconstruction. Our goal then is to keep the property of perfect reconstruction, while increasing the information rate and privacy threshold. One naive way to share more bits is to repeat the scheme several times, one for each bit. Of course, this does not help much in boosting the information rate. Our approach, on the other hand, is to use this naive repeated scheme to share a short random seed R. Suppose this gives us n parties with privacy threshold k_0. We then use R and the k-wise independent generator G mentioned above to generate an n-bit string Y, and use Y to share a secret X by computing $Y \oplus X$.

Note that now the length of the secret X can be as large as n and thus the information rate is increased to $1/2$. To reconstruct the secret, we can use the first n parties to

reconstruct R, then compute Y and finally X. Note that the whole computation can be done in $\mathcal{AC}^0$ since the k-wise independent generator G is computable in $\mathcal{AC}^0$. The privacy threshold, on the other hand, is the minimum of k_0 and k. This is because if an adversary learns nothing about R, then Y is k-wise independent and thus by looking at any k shares in $Y \oplus X$, the adversary learns nothing about X. This is the first step of our construction.

In the next step, we would like to boost the privacy threshold to $\Omega(n)$ while decreasing the information rate by at most a constant factor. Our approach for this purpose can be viewed as concatenating a larger outer protocol with a smaller inner protocol, which boosts the privacy threshold while keeping the information rate and the complexity of the whole protocol. More specifically, we first divide the parties obtained from the first step into small blocks, and then for each small block we use a good secret sharing scheme based on error correcting codes. Suppose the adversary gets to see a constant fraction of the shares, then on average for each small bock the adversary also gets to see only a constant fraction of the shares. Thus, by Markov's inequality and adjusting the parameters, the adversary only gets to learn the information from a constant fraction of the blocks. However, this is still not enough for us, since the outer protocol only has threshold $n^{\Omega(1)}$.

We solve this problem by using a threshold amplification technique. This is one of our main innovations, and a key step towards achieving both constant information rate and privacy threshold $\Omega(n)$ without sacrificing the error. On a high level, we turn the inner protocol itself into another concatenated protocol (i.e., a larger outer protocol combined with a smaller inner protocol), and then apply a random permutation. Specifically, we choose the size of the block mentioned above to be something

like $O(\log^2 n)$, apply a secret sharing scheme based on asymptotically good error correcting codes and obtain $O(\log^2 n)$ shares. We then divide these shares further into $O(\log n)$ smaller blocks each of size $O(\log n)$ (alternatively, this can be viewed as a secret sharing scheme using alphabet $\{0,1\}^{O(\log n)}$), and now we apply a random permutation of these smaller blocks. If we are to use a slightly larger alphabet, we can now store each block together with its index before the permutation as one share. Note that we need the index information when we try to reconstruct the secret, and the reconstruction can be done in $\mathcal{AC}^0$.

Now, suppose again that the adversary gets to see some small constant fraction of the final shares, then since we applied a random permutation, we can argue that each smaller block gets learned by the adversary only with some constant probability. Thus, in the larger block of size $O(\log^2 n)$, by a Chernoeff type bound, except with probability $1/\mathsf{poly}(n)$, we have that only some constant fraction of the shares are learned by the adversary. Note that here by using two levels of blocks, we have reduced the probability that the adversary learns some constant fraction of the shares from a constant to $1/\mathsf{poly}(n)$, which is much better for the outer protocol as we shall see soon. By adjusting the parameters we can ensure that the number of shares that the adversary may learn is below the privacy threshold of the larger block and thus the adversary actually learns nothing. Now, going back to the outer protocol, we know that the expected number of large blocks the adversary can learn is only $n/\mathsf{poly}(n)$; and again by a Chernoff type bound, except with probability $2^{-n^{\Omega(1)}}$, the outer protocol guarantees that the adversary learns nothing. This gives us a secret sharing scheme with privacy threshold $\Omega(n)$ while the information rate is still constant since we only increased the number of shares by a constant factor. With the $O(\log n)$ size alphabet,

we can actually achieve privacy threshold $(1-\alpha)n'$ for any constant $0 < \alpha < 1$, where n' is the total number of final parties.

To reduce to the binary alphabet, we can apply another secret sharing scheme based on error correcting codes to each share of length $O(\log n)$. In this case then we won't be able to achieve privacy threshold $(1-\alpha)n'$, but we can achieve $\beta n'$ for some constant $\beta > 0$. This is because if the adversary gets to see a small constant fraction of the shares, then by Markov's inequality only for some constant fraction of the smaller blocks the adversary can learn some useful information. Thus the previous argument still holds.

As described above, our general construction uses two levels of concatenated protocols, which corresponds to two levels of blocks. The first level has larger blocks of size $O(\log^2 n)$, where each larger block consists of $O(\log n)$ smaller blocks of size $O(\log n)$. We use this two-level structure to reduce the probability that an adversary can learn some constant fraction of shares, and this enables us to amplify the privacy threshold to $\Omega(n)$. We choose the smaller block to have size $O(\log n)$ so that both a share from the larger block with length $O(\log n)$ and its index information can be stored in a smaller block. This ensures that the information rate is still a constant even if we add the index information. Finally, the blocks in the second level are actually the blocks that go into the random permutation. This general strategy is one of our main contributions and we hope that it can find other applications.

The above construction gives an $\mathcal{AC}^0$ secret sharing scheme with good parameters. However, it is not a priori clear that it works for an adaptive adversary. In standard secret sharing schemes, a non-adaptive adversary and an adaptive adversary are almost equivalent since usually we have privacy error 0. More specifically, a secret sharing

scheme for a non-adaptive adversary with privacy error ϵ and privacy threshold k is also a secret sharing scheme for an adaptive adversary with privacy error $n^k\epsilon$ and privacy threshold k. However in our $\mathcal{AC}^0$ secret sharing scheme the error ϵ is not small enough to kill the n^k factor. Instead, we use the property of the random permutation to argue that our final distribution is essentially symmetric; and thus informally no matter how the adversary picks the shares to observe adaptively, he will not gain any advantage. This will show that our $\mathcal{AC}^0$ secret sharing scheme also works for an adaptive adversary.

To extend to robust secret sharing, we need to use robust secret sharing schemes instead of normal schemes for the first and second level of blocks. Here we use the nearly optimal robust secret sharing schemes based on various codes by Cheraghchi [Che16]. Unfortunately since we need to use it on a small block length of $O(\log n)$, the reconstruction error becomes $1/\mathsf{poly}(n)$. Another tricky issue here is that an adversary may modify some of the indices. Note that we need the correct index information in order to know which block is which before the random permutation. Suppose the adversary does not modify any of the indices, but only modify the shares, then the previous argument can go through exactly when we change the secret sharing schemes based on error correcting codes into robust secret sharing schemes. However, if the adversary modifies some indices, then we could run into situations where more than one block have the same index and thus we cannot tell which one is correct (and it's possible they are all wrong). To overcome this difficulty, we store every index multiple times among the blocks in the second level. Specifically, after we apply the random permutation, for every original index we randomly choose $O(\log n)$ blocks in the second level to store it. As the adversary can only corrupt a small constant fraction

of the blocks in the second level, for each such block, we can correctly recover its original index with probability $1 - 1/\mathsf{poly}(n)$ by taking the majority of the backups of its index. Thus by a union bound with probability $1 - 1/\mathsf{poly}(n)$ all original indices can be correctly recovered. In addition, we use the same randomness for each block to pick the $O(\log n)$ blocks, except we add a different shift to the selected blocks. This way, we can ensure that for each block the $O(\log n)$ blocks are randomly selected and thus the union bound still holds. Furthermore the randomness used here is also stored in every block in the second level, so that we can take the majority to reconstruct it correctly. In the above description, we sometimes need to take majority for n inputs, which is not computable in $\mathcal{AC}^0$. However, we note that by adjusting parameters we can ensure that at least say $2/3$ fraction of the inputs are the same, and in this case it suffices to take *approximate majority*, which can be computed in $\mathcal{AC}^0$ [Vio09].

For our error correcting codes, the construction is a simplified version of the robust secret sharing construction. Specifically, we first divide the message itself into blocks of the first level, and then encode every block using an asymptotically good code and divide the obtained codeword into blocks of the second level. Then we apply a random permutation to the blocks of the second level as before, and we encode every second level block by another asymptotically good code. In short, we replace the above mentioned robust secret sharing schemes by asymptotically good error correcting codes. We use the same strategy as in robust secret sharing to identify corrupted indices. Using a size of $O(\log^2 n)$ for blocks in the first level will result in decoding error $1/\mathsf{poly}(n)$, while using larger block size (e.g., $\mathsf{poly}\log(n)$) will result in decoding error $2^{-\mathsf{poly}\log(n)}$. This gives Theorem 2.1.8. To achieve even smaller error, we can first repeat each bit of the message r times for some parameter r. This

serves as an outer error correcting code, which can tolerate up to $r/3$ errors, and can be decoded in in $\mathcal{AC}^0$ by taking approximate majority. The two-level block structure and the argument we described before can now be used to show a smaller decoding error of $2^{-\Omega(r/\log^2 n)}$. This gives Theorem 2.1.9.

Comparison with related works on efficient error-correcting codes. As discussed above, our construction of error correcting codes shares some common ideas with earlier constructions of Smith [Smi07], Guruswami and Smith [GS16], and Hemenway et al. [Hem+11]. All these constructions have a common structure which has a "control-information" part and a "payload" part, where the payload part encodes the message using some randomness that is encoded in the control-information part. The general strategy to encode the payload part is to first encode the message, then do a random permutation over all the symbols, and finally encode again. The idea of using random permutations to randomize errors and thereby improve efficiency was also used by some earlier works such as [BBR88; Lip94]. Here we are using fully random permutations as in [BBR88], whereas [Smi07; GS16] used k-wise independent permutations and [Lip94; Hem+11] assumed the existence of a pseudorandom generator for permutations. Our goals are quite different from those considered in [Smi07; GS16; Hem+11]. The goal of [Smi07; GS16] was to optimize standard code parameters, so some components in their constructions are not local or in $\mathcal{AC}^0$. Hemenway et. al. [Hem+11] mainly focused on locally decodable codes in the computational secure setting so they use cryptographic primitives (based on cryptographic assumptions) such as semantically secure public-key encryption and pseudorandom generators. Their construction cannot be in $\mathcal{AC}^0$ unless they use stronger assumptions such that those cryptographic primitives can be computed in $\mathcal{AC}^0$. Our construction focuses

on the information theoretic secure setting. We observe that the payload part can be realized in $\mathcal{AC}^0$ if the encodings, before and after the permutation, are conducted over small blocks, though this only gives decoding error quasi-polynomially small. Finally, there is a technical difference between our encoding of the control information and the one used in prior works. Here we use a new index backup technique that be implemented (for both encoding and decoding) in $\mathcal{AC}^0$ and can be used to reconstruct all the indices under non-adaptive adversaries. The analysis for the backup technique argues that a non-adaptive adversary corrupting a small constant fraction of the shares can only corrupt at most $1/3$ fraction of the backups for one index, thus one can recover every index correctly by taking the approximate majority.

Secure broadcasting. We turn to describe the ideas behind our solution to the secure broadcasting problem from Section 2.1.3. Rather than use the naive approach of one-time pad, here a more clever solution is to use secret sharing (assuming that each party also has access to local private random bits). By first applying a secret sharing scheme to the input and then broadcasting the shares, a party can ensure that if the adversary only gets to see part of the messages (below the secrecy threshold), then the adversary learns nothing. In this case the parties do not even need shared secret key. However, one problem with this solution is that the adversary cannot be allowed to see more than $1/n$ fraction of the messages, since otherwise he can just choose the messages broadcasted from one particular party, and then the adversary learns the input of that party. This is the place where randomization comes into play. If in addition, we allow the parties to share a small number of secret random bits, then the parties can use this secret key to *randomly permute* the order in which the they broadcast their messages (after applying the secret sharing scheme). Since the

adversary does not know the secret key, we can argue that with high probability only a small fraction of each party's secret shares are observed. Therefore, by the properties of secret sharing we can say that the adversary learns almost nothing about each party's input. The crucial features of this solution solution are that first, the adversary can see some fixed fraction of messages, which is independent of the number of parties n (and thus can be much larger than $1/n$). Second, the number of shared secret random bits is much smaller than the naive approach of one-time pad. Indeed, as we show in Theorem 2.7.11, to achieve security parameter roughly r it is enough for the parties to share $O(r(\log n + \log r))$ random bits. Finally, by using an appropriate secret sharing scheme, the communication complexity of our protocol for each party is $O(m)$, which is optimal up to a constant factor. Note that here, by applying random permutation and allowing for a small probability of error, we simultaneously improve the security threshold (from $1/n$ to $\Omega(1)$) and the length of the shared secret key (from nm to $O(r(\log n + \log r))$).

2.1.5 Discussion and open problems.

In this chapter we continue the line of work on applying randomization and allowing a small failure probability for minimizing the computational complexity of cryptographic primitives and related combinatorial objects while maximizing the level of achievable security. In the context of secret sharing in $\mathcal{AC}^0$, we show how to get much better parameters by allowing an (exponentially) small privacy error. We note that achieving exponentially small error here is non-trivial. In fact, if we allow for a larger error then (for a non-adaptive adversary) there is a simple protocol for $\mathcal{AC}^0$ secret sharing: one can first take a random seed R of length $\Omega(n)$, and then apply

a deterministic $\mathcal{AC}^0$ extractor for bit-fixing sources to obtain an output Y of length $\Omega(n)$. The secret X can then be shared by computing the parity of Y and X. This way, one can still share $\Omega(n)$ bits of secret, and if the adversary only learns some small fraction of the seed, then the output Y is close to uniform by the property of the extractor, and thus X remains secret. However, by the lower bound of [CL16], the error of such $\mathcal{AC}^0$ extractors (or even for the stronger seeded $\mathcal{AC}^0$ extractors) is at least $2^{-\mathsf{poly}\log(n)}$. Therefore, one has to use additional techniques to achieve exponentially small error. We also extended our techniques to robust $\mathcal{AC}^0$ secret sharing schemes, stochastic error correcting codes for additive channels, and secure broadcasting. Several intriguing open problems remain.

First, in our robust $\mathcal{AC}^0$ secret sharing schemes, we only achieve reconstruction error $1/\mathsf{poly}(n)$. This is because we need to use existing robust secret sharing schemes on a block of size $O(\log n)$. Is it possible to avoid this and make the error exponentially small? Also, again in this case we can only handle non-adaptive adversaries, and it would be interesting to obtain a robust $\mathcal{AC}^0$ secret sharing scheme that can handle adaptive adversaries. These questions are open also for $\mathcal{AC}^0$ stochastic error correcting codes.

Second, as we mentioned in Remark 2.1.4 (see also [BW17]), a sufficiently small privacy error in an $\mathcal{AC}^0$ secret sharing scheme would imply an improved approximate degree lower bound for $\mathcal{AC}^0$ functions. Is it possible to improve our $\mathcal{AC}^0$ secret sharing scheme, and use this approach to obtain better approximate degree lower bound for $\mathcal{AC}^0$ functions? This seems like an interesting direction.

In addition, the privacy threshold amplification technique we developed, by using two levels of concatenated protocols together with a random permutation, is quite

general and we feel that it should have applications elsewhere. We note that the approach of combining an "outer scheme" with an "inner scheme" to obtain the best features of both has been applied in many previous contexts. For instance, it was used to construct better codes [Alo+92a; GS16] or better secure multi-party computation protocols [Dam+08]. However, in almost all of these previous applications, one starts with an outer scheme with a very good threshold (e.g., the Reed-Solomon code which has a large distance) and the goal is to use the inner scheme to inherit this good threshold while improving some other parameters (such as alphabet size). Thus, one only needs one level of concatenation. In our case, instead, we start with an outer scheme with a very weak threshold (e.g., the one-in-a-box function which only has privacy threshold $n^{1/3}$). By using two levels of concatenated protocols together with a random permutation, we can actually amplify this threshold to $\Omega(n)$ while simultaneously reducing the alphabet size. This is an important difference to previous constructions and one of our main contributions. We hope that these techniques can find other applications in similar situations.

Finally, since secret sharing schemes are building blocks of many other important cryptographic applications, it is an interesting question to see if the low-complexity secret sharing schemes we developed here can be used to reduce the computational complexity of other cryptographic primitives.

2.1.6 Chapter organization.

We introduce some notation and useful results in Section 2.2. In Section 2.3 we give our privacy threshold amplification techniques. In Section 2.4, we show how to increase the information rate using k-wise independent generators. Combining all the

above techniques, our final construction of $\mathcal{AC}^0$ secret sharing schemes is given in Section 2.5. Instantiations appear in Section 2.6. Finally, we give our constructions of robust $\mathcal{AC}^0$ secret sharing schemes, $\mathcal{AC}^0$ error correcting codes, and secure broadcast protocols in Section 2.7.

2.2 Preliminaries

Let $|\cdot|$ denote the size of the input set or the absolute value of an input real number, based on contexts.

For any set I of integers, for any $r \in \mathbb{Z}$, we denote $r + I$ or $I + r$ to be $\{i' : i' = i + r, i \in I\}$.

We use Σ to denote the alphabet. Readers can simply regard Σ as $\{0,1\}^l$ for some $l \in \mathbb{N}$. For $\sigma \in \Sigma$, let $\sigma^n = (\sigma, \sigma, \ldots, \sigma) \in \Sigma^n$. For any sequence $s = (s_1, s_2, \ldots, s_n) \in \Sigma^n$ and sequence of indices $W = (w_1, \ldots, w_t) \in [n]^t$ with $t \leq n$, let s_W be the subsequence $(s_{w_1}, s_{w_2}, \ldots, s_{w_t})$.

For any two sequences $a \in \Sigma^n, b \in \Sigma'^{n'}$ where $a = (a_1, a_2, \ldots, a_n), b = (b_1, b_2, \ldots, b_{n'})$, let $a \circ b = (a_1, \ldots, a_n, b_1, \ldots, b_{n'}) \in \Sigma^n \times \Sigma'^{n'}$.

Let $\mathsf{supp}(\cdot)$ denote the support of the input random variable. Let $I(\cdot)$ be the indicator function.

Definition 2.2.1 (Statistical Distance)**.** *The statistical distance between two random variables X and Y over Σ^n for some alphabet Σ, is $\mathsf{SD}(X,Y)$ which is defined as follows,*

$$\mathsf{SD}(X,Y) = 1/2 \sum_{a \in \Sigma^n} |\Pr[X = a] - \Pr[Y = a]|.$$

Here we also say that X is $\mathsf{SD}(X,Y)$-close to Y.

Lemma 2.2.2 (Folklore Properties of Statistical Distance [AB09])**.** *1. (Triangle Inequality) For any random variables X, Y, Z over Σ^n, we have*

$$\mathsf{SD}(X,Y) \leq \mathsf{SD}(X,Z) + \mathsf{SD}(Y,Z).$$

2. $\forall n, m \in \mathbb{N}$, any deterministic function $f : \{0,1\}^n \to \{0,1\}^m$ and any random variables X, Y over Σ^n, $\mathsf{SD}(f(X), f(Y)) \leq \mathsf{SD}(X,Y)$.

We will use the following well known perfect XOR secret sharing scheme.

Theorem 2.2.3 (Folklore XOR secret sharing)**.** *For any finite field $\mathbb{F}$, define $\mathsf{Share}_+ : \mathbb{F} \to \mathbb{F}^n$ and $\mathsf{Rec}_+ : \mathbb{F}^n \to \mathbb{F}$, such that for any secret $x \in \mathbb{F}$, $\mathsf{Share}_+(x) = y$ such that y is uniformly chosen in $\mathbb{F}^n$ conditioned on $\sum_{i\in[n]} y_i = x$ and Rec_+ is taking the sum of its input.*

$(\mathsf{Share}_+, \mathsf{Rec}_+)$ is an $(n, n-1)$ secret sharing scheme with share alphabet and message alphabet both being $\mathbb{F}$, message length 1, perfect privacy and reconstruction.

Definition 2.2.4 (Permutation)**.** *For any $n \in \mathbb{N}$, a permutation over $[n]$ is defined to be a bijective function $\pi : [n] \to [n]$.*

Definition 2.2.5 (k-wise independence)**.** *For any set S, let $X_1, \ldots, X_n$ be random variables over S. They are k-wise independent (and uniform) if any k of them are independent (and uniformly distributed).*

For any $r, n, k \in \mathbb{N}$, a function $g : \{0,1\}^r \to \Sigma^n$ is a k-wise (uniform) independent generator, if for $g(U) = (Y_1, \ldots, Y_n)$, $Y_1, \ldots, Y_n$ are k-wise independent (and uniform). Here U is the uniform distribution over $\{0,1\}^r$.

Definition 2.2.6 ([GUV09])**.** *A bipartite graph with N left vertices, M right vertices and left degree D is a (K, A) expander if for every set of left vertices $S \subseteq [N]$ of size*

K, we have $|\Gamma(S)| > AK$. It is a $(\leq K_{\max}, A)$ expander if it is a (K, A) expander for all $K \leq K_{\max}$.

Here $\forall x \in [N]$, $\Gamma(x)$ outputs the set of all neighbours of x. It is also a set function which is defined accordingly. Also $\forall x \in [N], d \in [D]$, the function $\Gamma : [N] \times [D] \to [M]$ is such that $\Gamma(x, d)$ is the dth neighbour of x.

Theorem 2.2.7 ([GUV09])**.** *For all constants $\alpha > 0$, for every $N \in \mathbb{N}$, $K_{\max} \leq N$, and $\epsilon > 0$, there exists an explicit $(\leq K_{\max}, (1-\epsilon)D)$ expander with N left vertices, M right vertices, left degree $D = O((\log N)(\log K_{\max})/\epsilon)^{1+1/\alpha}$ and $M \leq D^2 K_{\max}^{1+\alpha}$. Here D is a power of 2.*

Definition 2.2.8 ($\mathcal{AC}^0$)**.** *$\mathcal{AC}^0$ is the complexity class which consists of all families of circuits having constant depth and polynomial size. The gates in those circuits are* NOT, AND *and* OR*, where* AND *gates and* OR *gates have unbounded fan-in.*

For any circuit C, the size of C is denoted as $\mathsf{size}(C)$. The depth of C is denoted as $\mathsf{depth}(C)$. Usually when we talk about computations computable by $\mathcal{AC}^0$ circuits, we mean uniform $\mathcal{AC}^0$ circuits, if not stated specifically.

Lemma 2.2.9 (Forklore properties of $\mathcal{AC}^0$ circuits [AB09; Gol+07])**.** *The following are well known properties of $\mathcal{AC}^0$ circuits.*

For every $n \in \mathbb{N}$,

1. *([AB09] forklore) every boolean function $f : \{0,1\}^{l=\Theta(\log n)} \to \{0,1\}$ can be computed by an $\mathcal{AC}^0$ circuit of size $\mathsf{poly}(n)$ and depth 2.*

2. *([Gol+07]) for every $c \in \mathbb{N}$, every integer $l = \Theta(\log^c n)$, if the function $f_l : \{0,1\}^l \to \{0,1\}$ can be computed by a circuit with depth $O(\log l)$ and*

size $\mathsf{poly}(l)$*, then it can be computed by a circuit with depth* $c+1$ *and size* $\mathsf{poly}(n)$.

Remark 2.2.10. *We briefly describe the proof implied in [Gol+07] for the second property of our Lemma 2.2.9. As there exists an* $\mathcal{NC}^1$ *complete problem which is downward self-reducible, the function* f_l *can be reduced to (*$\mathcal{AC}^0$ *reduction) a function with input length* $O(\log n)$*. By Lemma 2.2.9 part 1, and noting that the reduction here is an* $\mathcal{AC}^0$ *reduction,* f_l *can be computed by an* $\mathcal{AC}^0$ *circuit.*

2.3 Random Permutation

2.3.1 Increasing the privacy threshold

The main technique we use here is random permutation.

Lemma 2.3.1 ([MV91; Hag91; Vio12])**.** *For any constant* $c \geq 1$*, there exists an explicit* $\mathcal{AC}^0$ *circuit* $C : \{0,1\}^r \to [n]^n$ *with size* $poly(n)$*, depth* $O(1)$ *and* $r = O(n^{c+1}\log n)$ *such that with probability* $1-2^{-n^c}$*,* $C(U_r)$ *gives a uniform random permutation of* $[n]$*; When this fails the outputs are not distinct.*

In the following we give a black box $\mathcal{AC}^0$ transformation of secret sharing schemes increasing the privacy threshold.

Construction 2.3.2. *For any* $n,k,m \in \mathbb{N}$ *with* $k \leq n$*, any alphabet* Σ,Σ_0*, let* $(\mathsf{Share},\mathsf{Rec})$ *be an* (n,k) *secret sharing scheme with share alphabet* Σ*, message alphabet* Σ_0*, message length* m*.*

Let $(\mathsf{Share}_+,\mathsf{Rec}_+)$ *be a* $(t,t-1)$ *secret sharing scheme with alphabet* Σ *by Theorem 2.2.3.*

For any constant $a \geq 1, \alpha > 0$, large enough $b \geq 1$, we can construct the following $(n' = tn\bar{n}, k' = (1-\alpha)n')$ secret sharing scheme $(\mathsf{Share}', \mathsf{Rec}')$ with share alphabet $\Sigma \times [n']$, message alphabet Σ_0, message length $m' = m\bar{n}$, where $t = O(\log n), \bar{n} = bn^{a-1}$.

Function $\mathsf{Share}' : \Sigma_0^{m'} \to (\Sigma \times [n'])^{n'}$ is as follows.

1. *On input secret $x \in \Sigma_0^{m\bar{n}}$, parse x to be $(x_1, x_2, \ldots, x_{\bar{n}}) \in (\Sigma_0^m)^{\bar{n}}$.*
2. *Compute $y = (y_1, \ldots, y_{\bar{n}}) = (\mathsf{Share}(x_1), \ldots, \mathsf{Share}(x_{\bar{n}}))$ and parse it to be $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_{n\bar{n}}) \in \Sigma^{n\bar{n}}$. Note that Share is from Σ_0^m to Σ^n.*
3. *Compute $(\mathsf{Share}_+(\hat{y}_1), \ldots, \mathsf{Share}_+(\hat{y}_{n\bar{n}})) \in (\Sigma^t)^{n\bar{n}}$ and split every entry to be t elements in Σ to get $y' = (y'_1, \ldots, y'_{n'}) \in \Sigma^{n'}$. Note that Share_+ is from Σ to Σ^t.*
4. *Generate π by Lemma 2.3.1 which is uniformly random over permutations of $[n']$. If it fails, which can be detected by checking element distinctness, set π to be such that $\forall i \in [n'], \pi(i) = i$.*
5. *Let*

$$\mathsf{Share}'(x) = (y'_{\pi^{-1}(1)} \circ \pi^{-1}(1), \ldots, y'_{\pi^{-1}(n')} \circ \pi^{-1}(n')) \in (\Sigma \times [n'])^{n'}.$$

Function $\mathsf{Rec}' : (\Sigma \times [n'])^{n'} \to \Sigma_0^{m'}$ is as follows.

1. *Parse the input to be $(y'_{\pi^{-1}(1)} \circ \pi^{-1}(1), \ldots, y'_{\pi^{-1}(n')} \circ \pi^{-1}(n'))$.*
2. *Compute $y' = (y'_1, \ldots, y'_{n'})$ according to the permutation.*
3. *Apply Rec_+ on y' for every successive t entries to get $\hat{y}$.*

4. *Parse $\hat{y}$ to be y.*

5. *Compute x by applying* Rec *on every entry of $\hat{y}$.*

6. *Output x.*

Lemma 2.3.3. *If* Share *and* Rec *can be computed by $\mathcal{AC}^0$ circuits, then* Share$'$ *and* Rec$'$ *can also be computed by $\mathcal{AC}^0$ circuits.*

Proof. As Share can be computed by an $\mathcal{AC}^0$ circuit, y can be computed by an $\mathcal{AC}^0$ circuit (uniform). By Lemma 2.2.9 part 1, we know that $(\mathsf{Share}_+, \mathsf{Rec}_+)$ both can be computed by $\mathcal{AC}^0$ circuits. By Lemma 2.3.1, $(\pi^{-1}(1), \pi^{-1}(2), \ldots, \pi^{-1}(n'))$ can be computed by an $\mathcal{AC}^0$ circuit. Also $\forall i \in [n'], y'_{\pi^{-1}(i)} = \bigvee_{j\in[n']}(y'_j \wedge (j = \pi^{-1}(i)))$. Thus Share$'$ can be computed by an $\mathcal{AC}^0$ circuit.

For Rec$'$, $\forall i \in [n'], y'_i = \bigvee_{j\in[n']}(y'_{\pi^{-1}(j)} \wedge (\pi^{-1}(j) = i))$. As Rec_+ can be computed by an $\mathcal{AC}^0$ circuit, y can be computed by an $\mathcal{AC}^0$ circuit. As Rec can be computed by an $\mathcal{AC}^0$ circuit, Rec$'$ can be computed by an $\mathcal{AC}^0$ circuit. □

Lemma 2.3.4. *If the reconstruction error of* $(\mathsf{Share}, \mathsf{Rec})$ *is η, then the reconstruction error of* $(\mathsf{Share}', \mathsf{Rec}')$ *is $\eta' = \bar{n}\eta$.*

Proof. According to the construction, as $(\mathsf{Share}_+, \mathsf{Rec}_+)$ has perfect reconstruction by Lemma 2.2.3, the y computed in Rec$'$ is exactly $(\mathsf{Share}(x_1), \mathsf{Share}(x_2), \ldots, \mathsf{Share}(x_{\bar{n}}))$. As $\forall i \in [\bar{n}], \Pr[\mathsf{Rec}(\mathsf{Share}(x_i)) = x_i] \geq 1 - \eta$,

$$\Pr[\mathsf{Rec}'(\mathsf{Share}'(x)) = x] = \Pr[\bigwedge_{i\in[\bar{n}]}(\mathsf{Rec}(\mathsf{Share}(x_i)) = x_i)] \geq 1 - \bar{n}\eta,$$

by the union bound. □

In order to show privacy, we need the following Chernoff Bound.

Definition 2.3.5 (Negative Correlation [AD11; Bis+16])**.** *Binary random variables* $X_1, X_2, \ldots, X_n$ *are negative correlated, if* $\forall I \subseteq [n]$,

$$\Pr[\bigwedge_{i \in I}(X_i = 1)] \leq \prod_{i \in I} \Pr[X_i = 1] \text{ and } \Pr[\bigwedge_{i \in I}(X_i = 0)] \leq \prod_{i \in I} \Pr[X_i = 0].$$

Theorem 2.3.6 (Negative Correlation Chernoff Bound [AD11; Bis+16])**.** *Let* $X_1, X_2, \ldots, X_n$ *be negatively correlated random variables with* $X = \sum_{i=1}^{n} X_i$, $\mu = \mathbb{E}[X]$.

- *For any* $\delta \in (0, 1)$,

$$\Pr[X \leq (1-\delta)\mu] \leq e^{-\delta^2 \mu/2} \text{ and } \Pr[X \geq (1+\delta)\mu] \leq e^{-\delta^2 \mu/3}.$$

- *For any* $d \geq 6\mu$, $\Pr[X \geq d] \leq 2^{-d}$.

Lemma 2.3.7. *Let* $\pi : [n] \to [n]$ *be a random permutation. For any set* $S, W \subseteq [n]$, *let* $u = \frac{|W|}{n}|S|$. *Then the following holds.*

- *for any constant* $\delta \in (0, 1)$,

$$\Pr[|\pi(S) \cap W| \leq (1-\delta)\mu] \leq e^{-\delta^2 \mu/2},$$

$$\Pr[|\pi(S) \cap W| \geq (1+\delta)\mu] \leq e^{-\delta^2 \mu/3}.$$

- *for any* $d \geq 6\mu$, $\Pr[|\pi(S) \cap W| \geq d] \leq 2^{-d}$.

Proof. For every $s \in S$, let X_s be the indicator such that $X_s = 1$ is the event that $\pi(s)$ is in W. Let $X = \sum_{s \in S} X_s$. So $|\pi(S) \cap W| = X$. Note that $\Pr[X_s = 1] = |W|/n$. So $\mu = \mathbb{E}(X) = \frac{|W|}{n}|S|$.

For any $I \subseteq S$,

$$\Pr[\bigwedge_{i \in I}(X_i = 1)] = \frac{|W|}{n} \cdot \frac{|W|-1}{n-1} \cdots \frac{|W|-|I|}{n-|I|}$$

(if $|W| < |I|$, it is 0). This is because the random permutation can be viewed as throwing elements $1, \ldots, n$ into n boxes uniformly one by one, where every box can have at most one element. We know that for $j = 1, \ldots, |I|$, $\frac{|W|-j}{n-j} \leq \frac{|W|}{n}$ as $|W| \leq n$. So $\Pr[\bigwedge_{i\in I}(X_i = 1)] \leq \prod_{i\in I} \Pr[X_i = 1]$. In the same way, for any $I \subseteq [n]$,

$$\Pr[\bigwedge_{i\in I}(X_i = 0)] = \frac{n-|W|}{n} \cdot \frac{n-|W|-1}{n-1} \cdots \frac{n-|W|-|I|}{n-|I|}$$

(if $n - |W| < |I|$, it is 0). Thus $\forall I \subseteq [n], \Pr[\bigwedge_{i\in I}(X_i = 0)] \leq \prod_{i\in I} \Pr[X_i = 0]$. By Theorem 2.3.6, the conclusion follows. □

We can get the following more general result by using Lemma 2.3.7.

Lemma 2.3.8. *Let $\pi : [n] \to [n]$ be a random permutation. For any $W \subseteq [n]$ with $|W| = \gamma n$, any constant $\delta \in (0,1)$, any $t, l \in \mathbb{N}^+$ such that $tl \leq \frac{0.9\delta}{1+0.9\delta}\gamma n$, any $\mathcal{S} = \{S_1, \ldots, S_l\}$ such that $\forall i \in [l], S_i \subseteq [n]$ are disjoint sets and $|S_i| = t$, let X_i be the indicator such that $X_i = 1$ is the event $|\pi(S_i) \cap W| \geq (1+\delta)\gamma t$. Let $X = \sum_{i\in[l]} X_i$. Then for any $d \geq 0$,*

$$\Pr[X \geq d] \leq e^{-2d+(e^2-1)e^{-\Omega(\gamma t)}l}.$$

Proof. For any $s > 0$, $\Pr[X \geq d] = \Pr[e^{sX} \geq e^{sd}] \leq \frac{\mathbb{E}[e^{sX}]}{e^{sd}}$ by Markov's inequality. For every $i \in [l]$, $\forall x_1, \ldots, x_{i-1} \in \{0,1\}$, consider $p = \Pr[X_i = 1 | \forall j < i, X_j = x_j]$. Let $\bar{S}_i = \bigcup_{j=1}^{i} S_j$ for $i \in [l]$. Note that the event $\forall j < i, X_j = x_j$ is the union of exclusive events $\pi(\bar{S}_{i-1}) = V, \forall j < i, X_j = x_j$ for $V \subseteq [n]$ with $|V| = (j-1)t$ and $\pi(\bar{S}_{i-1}) = V$ does not contradict $\forall j < i, X_j = x_j$. Conditioned on any one of those events, saying $\pi(\bar{S}_{i-1}) = V, \forall j < i, X_j = x_j$, π is a random bijective mapping from $[n] - \bar{S}_i$ to $[n] - V$. Note that $\frac{|W\cap([n]-V)|}{n-(i-1)t} \leq \frac{\gamma n}{n-\frac{0.9\delta}{1+0.9\delta}\gamma n} \leq \frac{\gamma n}{n-\frac{0.9\delta}{1+0.9\delta}n} \leq (1+0.9\delta)\gamma n$, since $(i-1)t \leq lt \leq \frac{0.9\delta}{1+0.9\delta}\gamma n$. So $\mathbb{E}[\pi(S_i) \cap W | | \pi(\bar{S}_{i-1}) =$

$V, \forall j < i, X_j = x_j] \leq (1 + 0.9\delta)\gamma t$. By Lemma 2.3.7, $\Pr[X_i = 1 | \pi(\bar{S}_{i-1}) = V, \forall j < i, X_j = x_j] = \Pr[|\pi(S_i) \cap W| \geq (1+\delta)\gamma t | \pi(\bar{S}_{i-1}) = V, \forall j < i, X_j = x_j] \leq e^{-\Omega(\gamma t)}$. Thus $p \leq e^{-\Omega(\gamma t)}$. Next note that

$$\mathbb{E}[e^{s\sum_{k=i}^{l} X_k} | \forall j < i, X_j = x_j]$$

$$= pe^s \mathbb{E}[e^{s\sum_{k=i+1}^{l} X_k} | \forall j < i, X_j = x_j, X_i = 1] + (1-p)\mathbb{E}[e^{s\sum_{k=i+1}^{l} X_k} | \forall j < i, X_j = x_j, X_i = 0]$$

$$\leq (pe^s + 1 - p) \max(\mathbb{E}[e^{s\sum_{k=i+1}^{l} X_k} | \forall j < i, X_j = x_j, X_i = 1], \mathbb{E}[e^{s\sum_{k=i+1}^{l} X_k} | \forall j < i, X_j = x_j, X_i = 0])$$

$$\leq e^{p(e^s-1)} \max(\mathbb{E}[e^{s\sum_{k=i+1}^{l} X_k} | \forall j < i, X_j = x_j, X_i = 1], \mathbb{E}[e^{s\sum_{k=i+1}^{l} X_k} | \forall j < i, X_j = x_j, X_i = 0])$$

$$\leq e^{e^{-\Omega(\gamma t)}(e^s-1)} \max(\mathbb{E}[e^{s\sum_{k=i+1}^{l} X_k} | \forall j < i, X_j = x_j, X_i = 1], \mathbb{E}[e^{s\sum_{k=i+1}^{l} X_k} | \forall j < i, X_j = x_j, X_i = 0]). \tag{2.1}$$

As this holds for every $i \in [l]$ and every $x_1, \ldots, x_{i-1} \in \{0,1\}$, we can iteratively apply the inequality and get the result that there exists $x'_1, \ldots, x'_l \in \{0,1\}$ such that $\mathbb{E}[e^{sX}] \leq e^{e^{-\Omega(\gamma t)}(e^s-1)}\mathbb{E}[e^{s\sum_{k=2}^{l} X_k} | X_1 = x'_1] \leq e^{2e^{-\Omega(\gamma t)}(e^s-1)}\mathbb{E}[e^{s\sum_{k=3}^{l} X_k} | X_1 = x'_1, X_2 = x'_2] \leq \cdots \leq e^{e^{-\Omega(\gamma t)}(e^s-1)l}$. Let's take $s = 2$. So $\Pr[X \geq d] \leq \frac{\mathbb{E}[e^{sX}]}{e^{sd}} \leq e^{-2d + (e^2-1)e^{-\Omega(\gamma t)}l}$.

□

Let's first show the non-adaptive privacy of this scheme.

Lemma 2.3.9. *If the non-adaptive privacy error of* $(\mathsf{Share}, \mathsf{Rec})$ *is* ϵ*, then the non-adaptive privacy error of* $(\mathsf{Share}', \mathsf{Rec}')$ *is* $\bar{n}(\epsilon + 2^{-\Omega(k)})$.

Proof. We show that there exists a distribution $\mathcal{D}$ such that for any string $x \in \Sigma_0^{m'}$, for any sequence of distinct indices $W = (w_1, w_2, \ldots, w_{k'}) \in [n']^{k'}$ (chosen before

observation),

$$\mathsf{SD}(\mathsf{Share}'(x)_W, \mathcal{D}) \leq \bar{n}(\epsilon + 2^{-\Omega(k)}).$$

For every $i \in [n\bar{n}]$, the block $\mathsf{Share}_+(\hat{y}_i)$ has length t. Let the indices of shares in $\mathsf{Share}_+(\hat{y}_i)$ be $S_i = \{(i-1)t+1, \ldots, it\}$.

For every $i \in [\bar{n}]$, let E_i be the event that for at most k of $j \in \{(i-1)n+1, \ldots, in\}$, $\pi(S_j) \subseteq W$. Let $E = \bigcap_{i\in[\bar{n}]} E_i$. We choose b to be such that $tn \leq \frac{0.9\alpha}{1+0.9\alpha}|W|$. So by Lemma 2.3.8, $\Pr[E_i] \geq 1 - e^{-\Omega(k)+(e^2-1)e^{-\Omega((1-\alpha)t)}n}$. We choose a large enough $t = O(\log n)$ such that $\Pr[E_i] \geq 1 - e^{-\Omega(k)}$. So $\Pr[E] \geq 1 - \bar{n}e^{-\Omega(k)}$ by the union bound.

Let's define the distribution $\mathcal{D}$ to be $\mathsf{Share}'(\sigma)_W$ for some $\sigma \in \Sigma_0^{m'}$. We claim that $\mathsf{Share}'(x)_W|E$ and $\mathcal{D}|E$ have statistical distance at most $\bar{n}\epsilon$. The reason is as follows.

Let's fix a permutation π for which E happens. We claim that $\mathsf{Share}'(x)_W$ is a deterministic function of at most k entries of each y_i for $i \in [\bar{n}]$ and some extra uniform random bits. This is because, as E happens, for those $i \in [n\bar{n}]$ with $\pi(S_i) \nsubseteq W$, the shares in $\pi(S_i) \cap W$ are independent of the secret by the privacy of $(\mathsf{Share}_+, \mathsf{Rec}_+)$. Note that they are also independent of other shares since the construction uses independent randomness for $\mathsf{Share}_+(\hat{y}_i), i \in [n\bar{n}]$. For those $i \in [n\bar{n}]$ with $\pi(S_i) \subseteq W$, the total number of them is at most k. So the claim holds. Hence by the privacy of $(\mathsf{Share}, \mathsf{Rec})$ with noting that $y_i, i \in [\bar{n}]$ are generated using independent randomness,

$$\mathsf{SD}(\mathsf{Share}'(x)_W, \mathcal{D}) \leq \bar{n}\epsilon.$$

So with probability at least $1 - \bar{n}e^{-\Omega(k)}$ over the fixing of π, $\mathsf{Share}'(x)_W$ and $\mathcal{D}$

have statistical distance at most $\bar{n}\epsilon$, which means that

$$\mathsf{SD}(\mathsf{Share}'(x)_W, \mathcal{D}) \leq \bar{n}(\epsilon + 2^{-\Omega(k)}).$$

□

Next we show the adaptive privacy.

Lemma 2.3.10. *For any alphabet Σ, any $n, k \in \mathbb{N}$ with $k \leq n$, for any distribution $X = (X_1, \ldots, X_n)$ over Σ^n, let $Y = ((X_{\pi^{-1}(1)} \circ \pi^{-1}(1)), \ldots, (X_{\pi^{-1}(n)} \circ \pi^{-1}(n)))$ where π is a random permutation over $[n] \to [n]$. For any adaptive observation W with $|W| = k$, Y_W is the same distribution as $Y_{[k]}$.*

Proof. Let $W = (w_1, \ldots, w_k)$.

We use induction.

For the base step, for any $x \in \Sigma$, any $i \in [n]$,

$$\Pr[Y_{w_1} = (x, i)] = \Pr[X_i = x]/n,$$

while

$$\Pr[Y_1 = (x, i)] = \Pr[X_i = x]/n.$$

So Y_{w_1} and Y_1 are the same distributions.

For the inductive step, assume that $Y_{W_{[i]}}$ and $Y_{[i]}$ are the same distributions. We know that for any $u \in (\Sigma \times [n])^i$,

$$\Pr[Y_{W_{[i]}} = u] = \Pr[Y_{[i]} = u].$$

Fix a $u \in (\Sigma \times [n])^i$. For any $v = (v_1, v_2) \in (\Sigma \times [n])$, where $v_1 \in \Sigma, v_2 \in [n]$, $\Pr[Y_{w_{i+1}} = v | Y_{W_{[i]}} = u] = 0$ if v_2 has already been observed in the previous i

observations; otherwise $\Pr[Y_{w_{i+1}} = v | Y_{W_{[i]}} = u] = \frac{\Pr[X_{v_2}=v_1]}{n-i}$. Also $\Pr[Y_{i+1} = v | Y_{[i]} = u] = 0$ if v_2 has already been observed in the previous i observations; otherwise $\Pr[Y_{i+1} = v | Y_{[i]} = u] = \frac{\Pr[X_{v_2}=v_1]}{n-i}$.

Thus $Y_{W_{[i+1]}}$ and $Y_{[i+1]}$ are the same distributions. This finishes the proof.

□

Lemma 2.3.11. *If* $(\mathsf{Share}, \mathsf{Rec})$ *has non-adaptive privacy error* ϵ*, then* $(\mathsf{Share}', \mathsf{Rec}')$ *has adaptive privacy error* $\bar{n}(\epsilon + 2^{-\Omega(k)})$.

Proof. First we assume that the adaptive observer always observes k' shares. For every observer M which does not observe k' shares, there exists another observer M' which can observe the same shares as M and then observe some more shares. That is to say that if the number of observed shares is less than k', M' will choose more unobserved shares (sequentially in a fixed order) to observe until k' shares are observed. Since we can use a deterministic function to throw away the extra observes of M' to get what M should observe, by Lemma 2.2.2 part 2, if the privacy holds for M' then the privacy holds for M. As a result, we always consider observers which observe k' shares.

By Lemma 2.3.10, for any $s \in \Sigma_0^{m'}$, $\mathsf{Share}'(s)_W$, for any adaptive observation W, is the same distribution as $\mathsf{Share}'(s)_{W'}$ where $W = \{w_1, w_2, \ldots, w_{k'}\}$, $W' = [k']$. As W' is actually a non-adaptive observation, by Lemma 2.3.9, for distinct $s, s' \in \{0,1\}^{m'}$, $\mathsf{SD}(\mathsf{Share}'(s)_{W'}, \mathsf{Share}'(s')_{W'}) \leq \bar{n}(\epsilon + 2^{-\Omega(k)})$. So

$$\mathsf{SD}(\mathsf{Share}'(s)_W, \mathsf{Share}'(s')_W) = \mathsf{SD}(\mathsf{Share}'(s)_{W'}, \mathsf{Share}'(s')_{W'}) \leq \bar{n}(\epsilon + 2^{-\Omega(k)}).$$

□

Theorem 2.3.12. *For any $n, m \in \mathbb{N}, m \leq n$, any $\epsilon, \eta \in [0,1]$ and any constant $a \geq 1, \alpha \in (0,1]$, if there exists an explicit (n,k) secret sharing scheme in $\mathcal{AC}^0$ with share alphabet Σ, message alphabet Σ_0, message length m, non-adaptive privacy error ϵ and reconstruction error η, then there exists an explicit $(n' = O(n^a \log n), (1-\alpha)n')$ secret sharing scheme in $\mathcal{AC}^0$ with share alphabet $\Sigma \times [n']$, message alphabet Σ_0, message length $\Omega(mn^{a-1})$, adaptive privacy error $O(n^{a-1}(\epsilon + 2^{-\Omega(k)}))$ and reconstruction error $O(n^{a-1}\eta)$.*

Proof. It immediately follows from Construction 2.3.2, Lemma 2.3.3, Lemma 2.3.4 and Lemma 2.3.11. □

2.3.2 Binary alphabet

In this subsection, we construct $\mathcal{AC}^0$ secret sharing schemes with binary alphabet based on some existing schemes with binary alphabets, enlarging the privacy threshold.

If we simply break each share in Construction 2.3.2 into bits, then we in fact get a secret sharing scheme with non-adaptive privacy. However, the privacy threshold becomes $O(n/\log n)$ which is sublinear, as the observer does not have to observe the indices. To overcome the barrier, we use some coding techniques and secret sharing for small blocks. An even bigger problem is that whether we can achieve adaptive privacy in this case. It seems to be hard since we have to break the indices into pieces. But surprisingly, we are still able to show adaptive privacy.

Lemma 2.3.13 ([Che+07] Section 4)**.** *For any $n \in \mathbb{N}$, any constant $\delta_0, \delta_1 \in (0,1)$, let $C \subseteq \mathbb{F}_2^n$ be an asymptotically good $(n, k = \delta_0 n, d = \delta_1 n)$ linear code.*

1. *There exists an (n,d) secret sharing scheme $(\mathsf{Share}, \mathsf{Rec})$ with alphabet $\{0,1\}$,*

message length k,perfect privacy and reconstruction. Here $\forall x \in \{0,1\}^k$, $\mathsf{Share}(x) = f(x) + c$ *with c drawn uniform randomly from* $C^\perp$ *(the dual code of C) and f is the encoding function from* $\{0,1\}^k$ *to C. For* $y \in \{0,1\}^n$, $\mathsf{Rec}(y)$ *is to find x such that there exists a* $c \in C^\perp$ *with* $f(x) + c = y$.

2. *For any* $p = \mathsf{poly}(n)$, *there exists an explicit* (n,d) *secret sharing scheme* $(\mathsf{Share}, \mathsf{Rec})$ *with alphabet* $\{0,1\}^p$, *message length k, perfect privacy and reconstruction.*

3. *If the codeword length is logarithmic (say* $n = O(\log N)$ *for some* $N \in \mathbb{N}$*), then both schemes can be constructed explicitly in* $\mathcal{AC}^0$ *(in N).*

Proof. The first assertion is proved in [Che+07].

The second assertion follows by applying the construction of the first assertion in parallel p times.

The third assertion holds because, when the codeword length is $O(\log N)$, both encoding and decoding functions have input length $O(\log N)$. For encoding, we can use any classic methods for generating asymptotically good binary codes. For decoding, we can try all possible messages to uniquely find the correct one. By Lemma 2.2.9, both functions can be computed by $\mathcal{AC}^0$ circuits.

□

Now we give the secret sharing scheme in $\mathcal{AC}^0$ with a constant privacy rate while having binary alphabet.

Construction 2.3.14. *For any* $n, k, m \in \mathbb{N}$ *with* $k, m \leq n$, *let* $(\mathsf{Share}, \mathsf{Rec})$ *be an* (n,k) *secret sharing scheme with alphabet* $\{0,1\}$, *message length m.*

Let $(\mathsf{Share}_C, \mathsf{Rec}_C)$ *be an* (n_C, k_C) *secret sharing scheme with alphabet* $\{0,1\}^{p=O(\log n)}$, *message length* m_C *by Lemma 2.3.13, where* $m_C = \delta_0 n_C$, $k_C = \delta_1 n_C$, $n_C = O(\log n)$ *for some constants* δ_0, δ_1.

Let $(\mathsf{Share}_0, \mathsf{Rec}_0)$ *be an* (n_0, k_0) *secret sharing scheme with alphabet* $\{0,1\}$, *message length* m_0 *by Lemma 2.3.13, where* $m_0 = \delta_0 n_0 = p + O(\log n)$, $k_0 = \delta_1 n_0$.

For any constant $a \geq 1$, *we can construct the following* $(n' = O(n^a), k' = \Omega(n'))$ *secret sharing scheme* $(\mathsf{Share}', \mathsf{Rec}')$ *with alphabet* $\{0,1\}$, *message length* $m' = m\bar{n}$, *where* $\bar{n} = \Theta(n^{a-1})$ *is large enough.*

Function $\mathsf{Share}' : \{0,1\}^{m'} \to \{0,1\}^{n'}$ *is as follows.*

1. *On input* $x \in \{0,1\}^{m\bar{n}}$, *parse it to be* $(x_1, x_2, \ldots, x_{\bar{n}}) \in (\{0,1\}^m)^{\bar{n}}$.

2. *Compute* $y = (y_1, \ldots, y_{\bar{n}}) = (\mathsf{Share}(x_1), \ldots, \mathsf{Share}(x_{\bar{n}})) \in (\{0,1\}^n)^{\bar{n}}$. *Split each entry to be blocks each has length* pm_C *to get* $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_{\tilde{n}}) \in (\{0,1\}^{pm_C})^{\tilde{n}}$, *where* $\tilde{n} = \bar{n}\lceil \frac{n}{pm_C} \rceil$.

3. *Let* $y^* = (\mathsf{Share}_C(\hat{y}_1), \ldots, \mathsf{Share}_C(\hat{y}_{\tilde{n}}))$. *Parse it to be* $y^* = (y^*_1, \ldots, y^*_{n^*}) \in (\{0,1\}^p)^{n^*}$, $n^* = \tilde{n} n_C$.

4. *Generate* π *by Lemma 2.3.1 which is uniform random over permutations of* $[n^*]$. *If it failed, which can be detected by checking element distinctness, set* π *to be such that* $\forall i \in [n^*], \pi(i) = i$.

5. *Compute*

$$z(x) = \mathsf{Share}'(x) = (\mathsf{Share}_0(y^*_{\pi^{-1}(1)} \circ \pi^{-1}(1)), \ldots, \mathsf{Share}_0(y^*_{\pi^{-1}(n^*)} \circ \pi^{-1}(n^*))) \in (\{0,1\}^{n_0}$$

6. *Parse* $z(x)$ *to be bits and output.*

Function $\mathsf{Rec}' : \{0,1\}^{n'=n_0 n^*} \to \{0,1\}^{m'}$ *is as follows.*

1. *Parse the input bits to be* $z \in (\{0,1\}^{n_0})^{n^*}$ *and compute*

$$(y^*_{\pi^{-1}(1)} \circ \pi^{-1}(1), \ldots, y^*_{\pi^{-1}(n^*)} \circ \pi^{-1}(n^*)) = (\mathsf{Rec}_0(z_1), \ldots, \mathsf{Rec}_0(z_{n^*})).$$

2. *Compute* $y^* = (y^*_1, \ldots, y^*_{n^*})$.

3. *Compute* $\hat{y}$ *by applying* Rec_C *on* y^* *for every successive* n_C *entries.*

4. *Parse* $\hat{y}$ *to be* y.

5. *Compute* x *by applying* Rec *on every entry of* y.

Lemma 2.3.15. *If* Share *and* Rec *can be computed by* $\mathcal{AC}^0$ *circuits, then* Share' *and* Rec' *can be computed by* $\mathcal{AC}^0$ *circuits.*

Proof. As Share can be computed by an $\mathcal{AC}^0$ circuit, y can be computed by an $\mathcal{AC}^0$ circuit. By Lemma 2.2.9 part 2 and 2.3.13, we know that $(\mathsf{Share}_C, \mathsf{Rec}_C)$ both can be computed by $\mathcal{AC}^0$ circuits. By Lemma 2.3.1, π can be computed by an $\mathcal{AC}^0$ circuit. Also $\forall i \in [n^*], y^*_{\pi^{-1}(i)} = \bigvee_{j \in [n^*]} (y^*_j \wedge (j = \pi^{-1}(i)))$. Thus Share' can be computed by an $\mathcal{AC}^0$ circuit.

For Rec', $\forall i \in [n^*], y^*_i = \bigvee_{j \in [n^*]} (y^*_{\pi^{-1}(j)} \wedge (\pi^{-1}(j) = i))$. As Rec_C can be computed by an $\mathcal{AC}^0$ circuit, y can be computed by an $\mathcal{AC}^0$ circuit. As Rec can be computed by an $\mathcal{AC}^0$ circuit, Rec' can be computed by an $\mathcal{AC}^0$ circuit. □

Lemma 2.3.16. *If the reconstruction error of* $(\mathsf{Share}, \mathsf{Rec})$ *is* η*, then the reconstruction error of* $(\mathsf{Share}', \mathsf{Rec}')$ *is* $\eta' = \bar{n}\eta$.

Proof. As $(\mathsf{Share}_0, \mathsf{Rec}_0)$ and $(\mathsf{Share}_C, \mathsf{Rec}_C)$ have perfect reconstruction by Lemma 2.3.13, the y computed in Rec' is exactly $(\mathsf{Share}(x_1), \mathsf{Share}(x_2), \ldots, \mathsf{Share}(x_{\bar{n}}))$. As $\forall i \in [\bar{n}], \Pr[\mathsf{Rec}(\mathsf{Share}(x_i)) = x_i] \geq 1 - \eta$,

$$\Pr[\mathsf{Rec}'(\mathsf{Share}'(x)) = x] = \Pr[\bigwedge_{i \in [\bar{n}]} (\mathsf{Rec}(\mathsf{Share}(x_i)) = x_i)] \geq 1 - \bar{n}\eta,$$

by the union bound. □

Lemma 2.3.17. *If the non-adaptive privacy error of* $(\mathsf{Share}, \mathsf{Rec})$ *is* ϵ*, then the non-adaptive privacy error of* $(\mathsf{Share}', \mathsf{Rec}')$ *is* $\bar{n}(\epsilon + 2^{-\Omega(k/\log^2 n)})$.

Proof. Let $k' = 0.9\delta_1^2 n'$. We show that there exists a distribution $\mathcal{D}$ such that for any string $x \in \{0,1\}^m$, for any $W \subseteq [n']$ with $|W| \leq k'$,

$$\mathsf{SD}(\mathsf{Share}'(x)_W, \mathcal{D}) \leq \bar{n}(\epsilon + 2^{-\Omega(k/\log^2 n)}).$$

Let $\mathcal{D}$ be $\mathsf{Share}'(\sigma)_W$ for some $\sigma \in \{0,1\}^{m'}$.

Consider an arbitrary observation $W \subseteq [n']$, with $|W| \leq k'$. Note that for at least $1 - 0.9\delta_1$ fraction of all blocks $z_i \in \{0,1\}^{n_0}, i = 1, \ldots, n^*$, at most δ_1 fraction of the bits in the block can be observed. Otherwise the number of observed bits is more than $0.9\delta_1 \times \delta_1 n'$. Let W^* be the index set of those blocks which have more than δ_1 fraction of bits being observed.

For every $i \in [n^*] \backslash W^*$, z_i is independent of $y^*_{\pi^{-1}(i)} \circ \pi^{-1}(i)$ by the privacy of $(\mathsf{Share}_0, \mathsf{Rec}_0)$. Note that z_i is also independent of $z_{i'}, i' \in [n^*], i' \neq i$ since it is independent of $y^*_{\pi^{-1}(i)} \circ \pi^{-1}(i)$ (its randomness is only from the randomness of the Share_0 function) and every Share_0 function uses independent randomness. So we only

have to show that

$$\mathsf{SD}(z_{W^*}(x), z_{W^*}(\sigma)) \leq \bar{n}(\epsilon + 2^{-\Omega(k/\log^2 n)}).$$

For every $i \in [\tilde{n}]$, let $S_i = \{(i-1)n_C + 1, \ldots, in_C\}$. Let X_i be the indicator that $|\pi(S_i) \cap W^*| > k_C, i \in [\tilde{n}]$. Note that $\mathbb{E}[|\pi(S_i) \cap W^*|] \leq 0.9\delta_1 n_C = 0.9k_C$.

For every $i \in [\bar{n}]$, let E_i be the event that $\sum_{j=(i-1)\lceil \frac{n}{pm_C} \rceil + 1}^{i\lceil \frac{n}{pm_C} \rceil} X_j \leq \frac{k}{pm_C}$. Let $E = \bigcap_{i \in [\bar{n}]} E_i$. We take $\bar{n}$ to be large enough such that $n_C \lceil \frac{n}{pm_C} \rceil \leq \frac{0.9 \times 0.1}{1 + 0.9 \times 0.1} |W^*|$. For every $i \in [\bar{n}]$, by Lemma 2.3.8,

$$1 - \Pr[E_i] \leq e^{-2k/(pm_C) + (e^2 - 1)e^{-\Omega(0.9\delta_1^2 n_C)} \lceil \frac{n}{pm_C} \rceil}.$$

We take $n_C = O(\log n)$ to be large enough such that the probability is at most $e^{-\Omega(k/(pm_C))} \leq e^{-\Omega(k/\log^2 n)}$.

Next we do a similar argument as that in the proof of Lemma 2.3.9. We know that $\Pr[E] \geq 1 - \bar{n}e^{-\Omega(k/\log^2 n)}$. We claim that $z_{W^*}(x)|E$ and $z_{W^*}(\sigma)|E$ have statistical distance at most $\bar{n}\epsilon$. The reason follows.

Let's fix a permutation π for which E happens. We claim that $z_{W^*}(x)$ is a deterministic function of at most k bits of each y_i for $i \in [\bar{n}]$ and some extra uniform random bits. This is because, as E happens, for those $i \in [\tilde{n}]$ with $|\pi(S_i) \cap W^*| \leq k_C$, the shares in $\pi(S_i) \cap W^*$ are independent of the secret by the privacy of $(\mathsf{Share}_C, \mathsf{Rec}_C)$. Note that they are also independent of other shares since the construction uses independent randomness for $\mathsf{Share}_C(\hat{y}_i), i \in [\tilde{n}]$. For those $i \in [\tilde{n}]$ with $|\pi(S_i) \cap W^*| > k_C$, the total number of them is at most $\frac{k}{pm_C}$. By the construction, $\mathsf{Share}'(x)_{W^*}$ is computed from at most $\frac{k}{pm_C} \times pm_C = k$ bits of each y_i for $i \in [\bar{n}]$ and some extra uniform random bits. Hence by the privacy of $(\mathsf{Share}, \mathsf{Rec})$ and noting that $y_i, \in [\bar{n}]$

are generated using independent randomness,

$$\mathsf{SD}(z_{W^*}(x), z_{W^*}(\sigma)) \leq \bar{n}\epsilon.$$

Thus with probability at least $1 - \bar{n}e^{-\Omega(k/\log^2 n)}$ over the fixing of π, $z_{W^*}(x)$ and $z_{W^*}(\sigma)$ have statistical distance at most $\bar{n}\epsilon$, which means that

$$\mathsf{SD}(z_{W^*}(x), z_{W^*}(\sigma)) \leq \bar{n}(\epsilon + e^{-\Omega(k/\log^2 n)}).$$

□

Lemma 2.3.18. *For any alphabet Σ, any $n \in \mathbb{N}$, Let $X = (X_1, \ldots, X_n)$ be an arbitrary distribution over Σ^n. For any $n_0, k_0 \in \mathbb{N}$ with $k_0 \leq n_0$, let $(\mathsf{Share}_0, \mathsf{Rec}_0)$ be an arbitrary (n_0, k_0)-secret sharing scheme with binary alphabet, message length $m_0 = \log|\Sigma| + O(\log n)$, perfect privacy. Let $Y = (\mathsf{Share}_0(X_{\pi^{-1}(1)} \circ \pi^{-1}(1)), \ldots, \mathsf{Share}_0(X_{\pi^{-1}(n)} \circ \pi^{-1}(n)))$ where π is a random permutation over $[n] \to [n]$. For any $t \leq n \cdot k_0$, let W be an any adaptive observation which observes t shares. Then there exists a deterministic function $f : \{0,1\}^{\mathsf{poly}(n)} \to \{0,1\}^t$ such that Y_W has the same distribution as $f(Y_{W'} \circ S)$, where S is uniform over $\{0,1\}^{\mathsf{poly}(n)}$ and $W' = [t'n_0], t' = \lceil \frac{t}{k_0} \rceil$.*

Proof. For every $i \in [n]$, Let $B_i = \{(i-1)n_0 + 1, \ldots, in_0\}$. Assume the adaptive adversary is M.

Let f be defined as the following.

Algorithm 2.3.1: $f(\cdot)$

Input : $y \in \{0,1\}^{t'n_0}$, $s \in \{0,1\}^{\mathsf{poly}(n)}$
Let $c = 1$;
$\forall i \in [n], l_i \in [n] \cup \{\mathsf{null}\}$ is assigned to be null;
Compute the secrets for the t' blocks y, which are

$$(x_1, \ldots, x_{t'}) \in (\{0,1\}^{m_0})^{t'};$$

Compute $(\mathsf{Share}_0(\sigma), \ldots, \mathsf{Share}_0(\sigma)) \in (\{0,1\}^{n_0})^n$ and parse it to be $r \in \{0,1\}^{n_0 n}$, for an arbitrary $\sigma \in \Sigma$. Here for each Share_0 function, we take some unused bits from s as the random bits used in that function.
Next f does the following computation by calling M;
while *M wants to observe the ith bit which is not observed previously* **do**
  Find $j \in [n]$ such that $i \in B_j$;
  if *the number of observed bits in the jth block is less than k_0* **then**
    Let M observe r_i;
  else
    Let I_j be the indices of the observed bits in the jth block. (The indices here are the relative indices in the jth block)
    if $l_j = \mathsf{null}$ **then**
      $l_j = c$;
      $c = c + 1$;
      Draw a string v^j from $\mathsf{Share}_0(x_c)|_{\mathsf{Share}_0(x_c)_{I_j} = r_{(j-1)n_0 + I_j}}$ by using some unused bits of s;
    end
    Let M observe $v^j_{i-(j-1)n_0}$;
  end
end

Let $W = (w_1, \ldots, w_t) \in [n \cdot n_0]^t$, $Z = f(Y_{W'} \circ S)$. Let $R \in \{0,1\}^{nn_0}$ be the random variable corresponds to r.

We use induction to show that Y_W has the same distribution as Z.

For the base case, the first bits of both random variables have the same distributions by the perfect privacy of $(\mathsf{Share}_0, \mathsf{Rec}_0)$.

For the inductive step, assume that, projected on the first d bits, the two distributions are the same. Fix the first d observed bits for both Y_W and Z to be $\bar{y} \in \{0,1\}^d$. Assume that the $(d+1)$th observation is to observe the w_dth bit where w_d is in B_j for some j.

If the number of observed bits in the jth block is less than k_0 then $Y_{\{w_1,\ldots,w_{d+1}\}\cap B_j}$ has the same distribution as $R_{\{w_1,\ldots,w_{d+1}\}\cap B_j}$, following the privacy of $(\mathsf{Share}_0, \mathsf{Rec}_0)$. Note that the blocks $Y_{\{w_1,\ldots,w_{d+1}\}\cap B_i}, i \in [n]$ are independent. The blocks $R_{\{w_1,\ldots,w_{d+1}\}\cap B_i}, i \in [n]$ are also independent. As f will output $R_{w_{d+1}}$, the conclusion holds for $d+1$.

Else, if the number of observed bits in the jth block is at least k_0, it is sufficient to show that $Y_{w_{d+1}}|_{Y_{\{w_1,\ldots,w_d\}}=\bar{y}}$ has the same distribution as that of $Z_{d+1}|_{Z_{\{1,\ldots,d\}}=\bar{y}}$. Note that there are c blocks such that W observes more than k_0 bits for each of them. Let $q_1,\ldots,q_c$ denote those blocks. Let $I = ((q_1-1)n_0 + I_{q_1},\ldots,(q_c-1)n_0 + I_{q_c})$, which is the set of indices of all observed bits. Note that $I \subseteq \{w_1,\ldots,w_d\}$.

By the privacy of the secret sharing scheme, for those blocks which have at most k_0 bits being observed, they are independent of the secret and hence independent of other blocks. So $Y_{w_{d+1}}|_{Y_{\{w_1,\ldots,w_d\}}=\bar{y}}$ is in fact $Y_{w_{d+1}}|_{Y_I=y^*}$ where y^* are the corresponding bits from $\bar{y}$ with a proper rearrangement according to I. From the definition of f we know that for $i \in [c]$, the observed bits in the q_ith block is exactly the same distribution as $(Y_{B_{l_{q_i}}})_{I_{q_i}} = \mathsf{Share}_0(x_{l_{q_i}})_{I_{q_i}}$. So for $Z_{d+1}|_{Z_{\{1,\ldots,d\}}=\bar{y}}$, it is the same distribution as

$$\begin{aligned} T &= (Y_{B_{l_j}})_{w_d-(j-1)n_0}|_{\bigwedge_{i=1}^{c}((Y_{B_{l_{q_i}}})_{I_{q_i}}=y^*_{(q_i-1)n_0+I_{q_i}})} \\ &= \mathsf{Share}_0(x_{l_j})_{w_d-(j-1)n_0}|_{\bigwedge_{i=1}^{c}(\mathsf{Share}_0(x_{l_{q_i}})_{I_{q_i}}=y^*_{(q_i-1)n_0+I_{q_i}})}. \end{aligned} \tag{2.2}$$

By Lemma 2.3.10, $(Y_{B_{q_1}},\ldots,Y_{B_{q_c}})$ has the same distribution as $(Y_{B_{l_{q_1}}},\ldots,Y_{B_{l_{q_c}}})$ as they both are the same distribution as $(\mathsf{Share}_0(x_1),\ldots,\mathsf{Share}_0(x_c))$. Thus $Y_{w_{d+1}}|_{Y_I=y^*}$

has the same distribution as T, as $Y_{w_{d+1}}|_{Y_I=y^*}$ is the distribution of some bits in $(Y_{B_{q_1}},\ldots,Y_{B_{q_c}})$ and T is the distribution of the corresponding bits (same indices) in $(Y_{B_{l_{q_1}}},\ldots,Y_{B_{l_{q_c}}})$. So we know that $Y_{w_{d+1}}|_{Y_{\{w_1,\ldots,w_d\}}=\bar{y}}$ has the same distribution as $Z_{d+1}|_{Z_{\{1,\ldots,d\}}=\bar{y}}$ and this shows our conclusion.

□

Lemma 2.3.19. *If the non-adaptive privacy error of* $(\mathsf{Share},\mathsf{Rec})$ *is* ϵ*, then the adaptive privacy error of* $(\mathsf{Share}',\mathsf{Rec}')$ *is* $\bar{n}(\epsilon+2^{-\Omega(k/\log^2 n)})$.

Proof. Let W be an adaptive observation . Let $W'=[\lceil |W|/k_0\rceil n_0]$. Let $|W|=\Omega(n')$ be small enough such that $|W'|\leq 0.9\delta_1^2 n'$. By Lemma 2.3.18, there exists a deterministic function f such that for any $x,x'\in\{0,1\}^{m'}$, $\mathsf{SD}(\mathsf{Share}'(x)_W,\mathsf{Share}(x')_W)=\mathsf{SD}(f(\mathsf{Share}'(x)_{W'}\circ S),f(\mathsf{Share}'(x')_{W'}\circ S))$ where S is the uniform distribution as defined in Lemma 2.3.18 which is independent of $\mathsf{Share}'(x)_{W'}$ or $\mathsf{Share}'(x')_{W'}$. By Lemma 2.2.2, we know that

$$\mathsf{SD}(f(\mathsf{Share}'(x)_{W'}\circ S),f(\mathsf{Share}'(x')_{W'}\circ S))\leq \mathsf{SD}(\mathsf{Share}'(x)_{W'},\mathsf{Share}'(x')_{W'}).$$

By Lemma 2.3.17 we know that

$$\mathsf{SD}(\mathsf{Share}'(x)_{W'},\mathsf{Share}'(x')_{W'})\leq \bar{n}(\epsilon+2^{-\Omega(k/\log^2 n)}).$$

Hence

$$\mathsf{SD}(\mathsf{Share}'(x)_{W},\mathsf{Share}'(x')_{W})\leq \bar{n}(\epsilon+2^{-\Omega(k/\log^2 n)}).$$

□

Theorem 2.3.20. *For any* $n,m\in\mathbb{N},m\leq n$*, any* $\epsilon,\eta\in[0,1]$ *and any constant* $a\geq 1$*, if there exists an explicit* (n,k) *secret sharing scheme in* $\mathcal{AC}^0$ *with alphabet*

$\{0,1\}$, *message length* m, *non-adaptive privacy error* ϵ *and reconstruction error* η, *then there exists an explicit* $(n' = O(n^a), k' = \Omega(n'))$ *secret sharing scheme in* $\mathcal{AC}^0$ *with alphabet* $\{0,1\}$, *message length* $\Omega(mn^{a-1})$, *adaptive privacy error* $O(n^{a-1}(\epsilon + 2^{-\Omega(k/\log^2 n)}))$ *and reconstruction error* $O(n^{a-1}\eta)$.

Proof. It follows from Construction 2.3.14, Lemma 2.3.15, 2.3.16 and 2.3.19.

□

2.4 k-wise independent generator in AC0

In this section we focus on increasing the secret length to be linear of the number of shares while keeping the construction in $\mathcal{AC}^0$. The privacy rate is not as good as the previous section. The main technique is to use the following well known k-wise independent generator which is constructed from expander graphs.

Theorem 2.4.1 ([MST06])**.** *For any* $N, D, M \in \mathbb{N}$, *any* $\epsilon > 0$, *if there exists a* $(\leq K_{\max}, (\frac{1}{2} + \epsilon)D)$ *expander with left set of vertices* $[N]$, *right set of vertices* $[M]$, *left degree* D, *then the function* $g : \{0,1\}^M \to \{0,1\}^N$, *defined by* $g(x)_i = \bigoplus_{j\in[D]} x_{\Gamma(i,j)}, i = 1, 2, \ldots, N$, *is a* $K_{\max}$*-wise uniform independent generator.*

Proof. For any subset $S \subseteq [N]$ with $|S| \leq K_{\max}$, there exists a $u \in \Gamma(S)$ such that $\exists v \in S, u \in \Gamma(v)$ while $\forall w \in S$ with $w \neq v$, $u \notin \Gamma(w)$. This is because if not, then $|\Gamma(S)| \leq \frac{1}{2}D|S|$ which contradicts that Γ is a $(\leq K_{\max}, (\frac{1}{2} + \epsilon)D)$ expander.

As

$$\bigoplus_{i\in S} g(x)_i = \bigoplus_{i\in S} \bigoplus_{j\in[D]} x_{\Gamma(i,j)},$$

$\bigoplus_{i\in S} g(U_M)_i$ is uniform.

By the Information Theoretic XOR-Lemma of [Gol95], for every set $S' \subseteq [N]$ of size $K_{\max}$, $g(U_M)_{S'}$ is uniform. Thus g is a $K_{\max}$-wise uniform independent generator. □

Theorem 2.4.2. *For any $M \in \mathbb{N}$, $N = \mathsf{poly}(M)$, any alphabets Σ_0, Σ, any constant $\gamma \in (0,1]$, there exists an explicit K-wise independent generator $g : \Sigma_0^M \to \Sigma^N$ in $\mathcal{AC}^0$, where $K = (\frac{M \log |\Sigma_0|}{\log |\Sigma|})^{1-\gamma}$.*

Proof. We first consider $\Sigma_0 = \Sigma = \{0,1\}$. By Theorem 2.2.7 for any constant α and every $\epsilon > 0$ there exists an explicit function $\Gamma : [N] \times [D] \to [M]$ which is the neighbour function of a $(\leq K_{\max}, (\frac{1}{2}+\epsilon)D)$ expander, where $D = O((\log N)(\log K_{\max})/\epsilon)^{1+1/\alpha}$ and $M \leq D^2 K_{\max}^{1+\alpha}$. We take $\epsilon = 0.1$ and take α to be a small enough constant such that $K_{\max} \geq M^{1-\gamma}$. By Theorem 2.4.1 we get an explicit K-wise independent generator $g : \{0,1\}^M \to \{0,1\}^N$ where $K = M^{1-\gamma}$.

For arbitrary alphabets Σ_0, Σ, we simply apply the generator for $\log |\Sigma|$ times in parallel using independent seeds. Note that the total seed length is $M \log |\Sigma_0|$ by parsing the input symbols into bits. So for every one of the $|\Sigma|$ generator in parallel, its seed length is $\frac{M \log |\Sigma_0|}{\log |\Sigma|}$. Hence each of them is a K-wise uniform independent generator with $K = (\frac{M \log |\Sigma_0|}{\log |\Sigma|})^{1-\gamma}$.

By Theorem 2.4.1, the construction take XOR over D bits. So by Lemma 2.2.9, it can be computed in $\mathcal{AC}^0$. Thus the generator can be computed in $\mathcal{AC}^0$.

□

Now we give the construction of secret sharing schemes in $\mathcal{AC}^0$ with large message rate (saying $1 - 1/\mathsf{poly}(n)$).

Construction 2.4.3. *For any $n, k, m \in \mathbb{N}$ with $k \leq n$, any alphabets Σ_0, Σ, let $(\mathsf{Share}, \mathsf{Rec})$ be an (n, k) secret sharing scheme with share alphabet Σ, message alphabet Σ_0, message length m.*

For any constant $a > 1, \gamma \in (0, 1]$, we construct the following $(n' = n + m', k' = \min(k, l))$ secret sharing scheme $(\mathsf{Share}', \mathsf{Rec}')$ with alphabet Σ, message length $m' = \Omega(n^a)$, where $l = \Theta(\frac{m \log |\Sigma_0|}{\log |\Sigma|})^{1-\gamma}$.

The function $\mathsf{Share}' : \Sigma^{m'} \to \Sigma^{n'}$ is as follows.

1. *Let $g_\Gamma : \Sigma_0^m \to \Sigma^{m'}$ be the l-wise independent generator by Theorem 2.4.2.*
2. *For secret $x \in \Sigma^{m'}$, we draw r uniform randomly from Σ_0^m let*

$$\mathsf{Share}'(x) = (\mathsf{Share}(r), g_\Gamma(r) \oplus x).$$

The function $\mathsf{Rec}' : \Sigma^{n'} \to \Sigma^{m'}$ is as follows.

1. *The input is $y = (y_1, y_2)$ where $y_1 \in \Sigma^n, y_2 \in \Sigma^{m'}$.*
2. *Let*

$$\mathsf{Rec}'(y) = g_\Gamma(\mathsf{Rec}(y_1)) \oplus y_2.$$

Lemma 2.4.4. *If Share and Rec can be computed by $\mathcal{AC}^0$ circuits, then Share' and Rec' can be computed by $\mathcal{AC}^0$ circuits.*

Proof. As Share can be computed by an $\mathcal{AC}^0$ circuit and $g_\Gamma(r) \oplus x$ can be computed by a CNF or DNF, Share' can be computed by an $\mathcal{AC}^0$ circuit.

Similarly, Rec' can also be computed by an $\mathcal{AC}^0$ circuit.

□

Lemma 2.4.5. *If the reconstruction error of* $(\mathsf{Share}, \mathsf{Rec})$ *is* η*, then the reconstruction error of* $(\mathsf{Share}', \mathsf{Rec}')$ *is* $\eta' = \eta$.

Proof. Let the input for Rec' be $(y_1, y_2) = (\mathsf{Share}(r), g_\Gamma(r) \oplus x)$. If $\mathsf{Rec}(\cdot)$ computes correctly then

$$\mathsf{Rec}'(y) = g_\Gamma(\mathsf{Rec}(y_1)) \oplus y_2 = g_\Gamma(r) \oplus g_\Gamma(r) \oplus x = x,$$

which means that Rec' recovers the correct secret.

So $\eta' = \eta$.

□

Next we show the non-adaptive privacy error of the construction.

Lemma 2.4.6. *If the non-adaptive privacy error of* $(\mathsf{Share}, \mathsf{Rec})$ *is* ϵ*, then the non-adaptive privacy error of* $(\mathsf{Share}', \mathsf{Rec}')$ *is also* ϵ.

Proof. Consider an arbitrary set $W \subseteq [n']$ of size k'. We view W as the union of two disjoint sets $W_1 \subseteq [n]$ and $W_2 \subseteq \{n+1, \ldots, n+m'\}$. Consider any two distinct secrets $x, x' \in \Sigma^{m'}$. As g_Γ is an l-wise independent generator and $k' \leq l$, $\mathsf{Share}'(x)_{W_2} = (g_\Gamma(R) \oplus x)_{W_2}$ and $\mathsf{Share}'(x')_{W_2} = (g_\Gamma(R) \oplus x')_{W_2}$ are both uniform distributions where R is uniform over Σ^m. For any string $u \in \Sigma^{m'}$, the statistical distance between the distribution $\mathsf{Share}'(x)_{W_1}|_{\mathsf{Share}'(x)_{W_2}=u}$ and the distribution $\mathsf{Share}'(x')_{W_1}|_{\mathsf{Share}'(x')_{W_2}=u}$ is ϵ, because $(\mathsf{Share}, \mathsf{Rec})$ has privacy error ϵ.

So we have that

$$\begin{aligned}
&\mathsf{SD}(\mathsf{Share}'(x)_W, \mathsf{Share}'(x')_W) \\
=&\sum_{u\in\Sigma^{m'}} \frac{1}{|\Sigma|^{m'}} \mathsf{SD}(\mathsf{Share}'(x)_{W_1}|_{\mathsf{Share}'(x)_{W_2}=u}, \mathsf{Share}'(x')_{W_1}|_{\mathsf{Share}'(x')_{W_2}=u}) \qquad (2.3) \\
=&\epsilon.
\end{aligned}$$

□

Theorem 2.4.7. *For any $n, m \in \mathbb{N}, m \leq n$, any $\epsilon, \eta \in [0,1]$, any constant $\gamma \in (0,1]$, any $m' = \mathsf{poly}(n)$ and any alphabets Σ_0, Σ, if there exists an explicit (n,k) secret sharing scheme in $\mathcal{AC}^0$ with share alphabet Σ, message alphabet Σ_0, message length m, non-adaptive privacy error ϵ and reconstruction error η, then there exists an explicit $(n+m', \min(k, (\frac{m\log|\Sigma_0|}{\log|\Sigma|})^{1-\gamma}))$ secret sharing scheme in $\mathcal{AC}^0$ with alphabet Σ, message length m', non-adaptive privacy error ϵ and reconstruction error η.*

Proof. It immediately follows from Construction 2.4.3, Lemma 2.4.4, 2.4.5 and 2.4.6.

□

2.5 Final construction

In this section we give our final $\mathcal{AC}^0$ construction of secret sharing schemes which has constant message rate and constant privacy rate.

Our construction will use both random permutation and k-wise independent generator proposed in the previous sections.

2.5.1 The construction

We first give the construction with a relatively big alphabet.

Construction 2.5.1. *For any $n, k, m \in \mathbb{N}$ with $k, m \leq n$, any alphabets Σ_0, Σ, let $(\mathsf{Share}, \mathsf{Rec})$ be an (n, k) secret sharing scheme with share alphabet Σ, message alphabet Σ_0, message length m.*

Let $(\mathsf{Share}_C, \mathsf{Rec}_C)$ be an (n_C, k_C) secret sharing scheme from Lemma 2.3.13 with alphabet Σ, message length m_C, where $m_C = \delta_0 n_C$, $k_C = \delta_1 n_C$, $n_C = O(\log n)$ for some constant δ_0, δ_1.

For any constant $a \geq 1$, $\gamma \in (0, 1]$, we can construct the following $(n' = O(n^a), k' = \Omega(n')$ secret sharing scheme $(\mathsf{Share}', \mathsf{Rec}')$ with share alphabet $\Sigma \times [n']$, message alphabet Σ, message length $m' = \Omega(n')$.

The function $\mathsf{Share}' : \Sigma^{m'} \to (\Sigma \times [n'])^{n'}$ is as follows.

1. *Let $\bar{n} = \Theta(n^{a-1})$ where the constant factor is large enough.*
2. *Let $g_\Gamma : \Sigma_0^{m\bar{n}} \to \Sigma^{m'}$ be the l-wise independent generator by Theorem 2.4.2, where $l = \Omega(\frac{m\bar{n} \log |\Sigma_0|}{\log |\Sigma|})^{1-\gamma}$.*
3. *For secret $x \in \Sigma^{m'}$, we draw a string $r = (r_1, \ldots, r_{\bar{n}})$ uniformly from $(\Sigma_0^m)^{\bar{n}}$.*
4. *Let $y = (y_s, y_g)$, where $y_s = (\mathsf{Share}(r_1), \ldots, \mathsf{Share}(r_{\bar{n}})) \in (\Sigma^n)^{\bar{n}}$ and $y_g = g_\Gamma(r) \oplus x \in \Sigma^{m'}$.*
5. *Get $\hat{y}_s \in (\Sigma^{m_C})^{n_s}$ from y_s by parsing $y_{s,i}$ to be blocks each having length m_C for every $i \in [\bar{n}]$, where $n_s = \lceil \frac{n}{m_C} \rceil \bar{n}$.*
6. *Get $\hat{y}_g \in (\Sigma^{m_C})^{n_g}$ from y_g by parsing y_g to be blocks each having length m_C, where $n_g = \lceil \frac{m'}{m_C} \rceil$.*

7. *Compute*

$$(\mathsf{Share}_C(\hat{y}_{s,1}), \ldots, \mathsf{Share}_C(\hat{y}_{s,n_s}), \mathsf{Share}_C(\hat{y}_{g,1}), \ldots, \mathsf{Share}'_C(\hat{y}_{g,n_g})).$$

and parse it to be $y' = (y'_1, \ldots, y'_{n'}) \in \Sigma^{n'}$, *where* $n' = (n_s + n_g)n_C$.

8. *Generate a random permutation* $\pi : [n'] \to [n']$ *and output*

$$z = ((y'_{\pi^{-1}(1)} \circ \pi^{-1}(1)), (y'_{\pi^{-1}(2)} \circ \pi^{-1}(2)), \ldots, (y'_{\pi^{-1}(n')} \circ \pi^{-1}(n'))) \in (\Sigma \times [n'])^{n'}.$$

The function $\mathsf{Rec}' : (\Sigma \times [n'])^{n'} \to \Sigma^{m'}$ *is as follows.*

1. *The input is* $z = ((y'_{\pi^{-1}(1)} \circ \pi^{-1}(1)), (y'_{\pi^{-1}(2)} \circ \pi^{-1}(2)), \ldots, (y'_{\pi^{-1}(n')} \circ \pi^{-1}(n')))$.
2. *Compute* $y' = (y'_1, \ldots, y'_{n'})$.
3. *Parse* y' *to be* (y'_s, y'_g) *where* $y'_s = (y'_{s,1}, \ldots, y'_{s,n_s}) \in (\Sigma^{n_C})^{n_s}$, $y'_g = (y'_{g,1}, \ldots, y'_{g,n_g}) \in (\Sigma^{n_C})^{n_g}$.
4. *Compute* $(\mathsf{Rec}_C(y'_{s,1}), \ldots, \mathsf{Rec}_C(y'_{s,n_g}))$ *and* $(\mathsf{Rec}_C(y'_{g,1}), \ldots, \mathsf{Rec}_C(y'_{g,n_g}))$. *Parse them to get* y_s *and* y_g.
5. *Compute* r *by applying* Rec *on every entry of* y_s.
6. *Output*

$$\mathsf{Rec}'(z) = g_\Gamma(r) \oplus y_g.$$

Lemma 2.5.2. *If* $(\mathsf{Share}, \mathsf{Rec})$ *can be computed by* $\mathcal{AC}^0$ *circuits, then* $(\mathsf{Share}', \mathsf{Rec}')$ *can be computed by* $\mathcal{AC}^0$ *circuits.*

Proof. By Theorem 2.4.2, g_Γ can be computed by an $\mathcal{AC}^0$ circuit. As Share can be computed by an $\mathcal{AC}^0$ circuit, y can be computed by an $\mathcal{AC}^0$ circuit. By Lemma 2.2.9 part 2, as $n_C = O(\log n)$, $(\mathsf{Share}_C, \mathsf{Rec}_C)$ can be computed by an $\mathcal{AC}^0$ circuit. By Lemma 2.3.1 the random permutation π can be computed by an $\mathcal{AC}^0$ circuit. Also $\forall i \in [n'], y'_{\pi^{-1}(i)} = \bigvee_{j\in[n']}(y'_j \wedge (j = \pi^{-1}(i)))$. Thus Share' can be computed by an $\mathcal{AC}^0$ circuit.

For Rec', $\forall i \in [n'], y'_i = \bigvee_{j\in[n']}(y'_{\pi^{-1}(j)} \wedge (\pi^{-1}(j) = i))$. As Rec_C and Rec can be computed by an $\mathcal{AC}^0$ circuits, Rec' can be computed by an $\mathcal{AC}^0$ circuit.

□

Lemma 2.5.3. *If the reconstruction error of* $(\mathsf{Share}, \mathsf{Rec})$ *is* η*, then the reconstruction error of* $(\mathsf{Share}', \mathsf{Rec}')$ *is* $\eta' = \bar{n}\eta$.

Proof. As $(\mathsf{Share}_C, \mathsf{Rec}_C)$ has perfect reconstruction, the error only occurs when we apply Rec. So we can compute each $r_i, i = 1, \ldots, \bar{n}$ correctly except with error η. By the union bound, with probability $1 - \bar{n}\eta$, r is correctly computed. Once we can compute r correctly, the secret $x = g_\Gamma(r) \oplus y_g$. Note that the correctness of y_g is guaranteed. This is because from z we can get the value of every entry of y' since each entry of z includes one entry of y' and an index showing which entry of y' it is. So y_g is correct as it is part of y'.

□

Lemma 2.5.4. *If the non-adaptive privacy error of* $(\mathsf{Share}, \mathsf{Rec})$ *is* ϵ*, then the non-adaptive privacy error of* $(\mathsf{Share}', \mathsf{Rec}')$ *is* $\bar{n}(\epsilon + e^{-\Omega(k/\log n)}) + e^{-\Omega(l/\log n)}$.

Proof. Let $k' = 0.9\delta_1 n'$. Consider an arbitrary $W \subseteq [n']$ with $|W| \leq k'$.

For every $i \in [n_s]$, let $S_i = \{n_C(i-1)+1, \ldots, n_C i\}$. Let $\mathcal{S} = \{S_1, \ldots, S_{n_s}\}$. Let X_i be the indicator such that $X_i = 1$ is the event $|\pi(S_i) \cap W| > k_C$. Note that $\mathbb{E}[|\pi(S_i) \cap W|] \leq 0.9\delta_1 n_C = 0.9k_C$.

For every $i \in [\bar{n}]$, let E_i be the event that $\sum_{j=(i-1)\lceil \frac{n}{m_C} \rceil + 1}^{i\lceil \frac{n}{m_C} \rceil} X_j \leq \frac{k}{m_C}$. Since $\bar{n}$ is large enough, $n_C \lceil \frac{n}{m_C} \rceil \leq \frac{0.9 \times 0.1}{1+0.9\times 0.1}|W|$. By Lemma 2.3.8,

$$1 - \Pr[E_i] \leq e^{-2k/m_C + (e^2-1)e^{-\Omega(0.9\delta_1 n_C)}\lceil \frac{n}{m_C} \rceil}.$$

We take $n_C = O(\log n)$ to be large enough such that the probability is at most $e^{-\Omega(k/m_C)} \leq e^{-\Omega(k/\log n)}$.

Let $\mathcal{T} = \{T_1, \ldots, T_{n_g}\}$, where $T_i = n_s n_C + \{n_C(i-1)+1, \ldots, n_C i\}, i = 1, \ldots, n_g$. Let Y_i be the indicator such that $Y_i = 1$ is the event $|\pi(T_i) \cap W| > k_C$. Let $Y = \sum_{i \in [n_g]} Y_i$. Note that $\mathbb{E}[|\pi(T_i) \cap W|] \leq 0.9\delta_1 n_C = 0.9k_C$. Let E_g be the event that $Y \leq \frac{l}{m_C}$. Since $\bar{n}$ is large enough, we can have $n_C n_g \leq \frac{0.9\times 0.1}{1+0.9\times 0.1}|W|$ when $m' = \Omega(n)$. By Lemma 2.3.8,

$$1 - \Pr[E_g] \leq e^{-2l/m_C + (e^2-1)e^{-\Omega(0.9\delta_1 n_C)} n_g}.$$

We take $n_C = O(\log n)$ to be large enough such that the probability is at most $e^{-\Omega(l/m_C)} \leq e^{-\Omega(l/\log n)}$.

Let E be the event that $(\bigcap_{i\in[\bar{n}]} E_i) \cap E_g$. By the union bound, $\Pr[E] \geq 1 - \bar{n}e^{-\Omega(k/\log n)} - e^{-\Omega(l/\log n)}$. We claim that $\mathsf{Share}'(x)_W|E$ and $\mathsf{Share}'(\sigma)_W|E$ have statistical distance at most $\bar{n}\epsilon$, where σ is an arbitrary string in $\Sigma^{m'}$. The reason is as follows.

We fix a permutation π for which E happens. Let $W_s = (\bigcup_{i\in[n_s]} S_i) \cap W$, $W_g = (\bigcup_{i\in[n_g]} T_i) \cap W$. Let R be the random variable which corresponds to the

random choice of r.

We claim that $\mathsf{Share}'(x)_{W_g}$ is a deterministic function of at most l entries of y_g and some extra uniform random bits. As E_g happens, for those $i \in [n_g]$ with $|\pi(T_i) \cap W| \leq k_C$, the shares indexed by $\pi(T_i) \cap W$ are independent of the secret by the privacy of $(\mathsf{Share}_C, \mathsf{Rec}_C)$. Note that they are also independent of other shares since the construction uses independent randomness for sharing $\hat{y}_{g,i}, i \in [n_g]$ and $\hat{y}_{s,i}, i \in [n_s]$. For those $i \in [n_g]$ with $|\pi(T_i) \cap W| > k_C$, the total number of them is at most $\frac{l}{m_C}$. So $\mathsf{Share}'(x)_{W_g}$ is computed from at most $\frac{l}{m_C} \times m_C = l$ entries of y_g and some extra uniform random bits.

As $g_\Gamma(\cdot)$ is an l-wise independent generator, the distribution of $\mathsf{Share}'(x)_{W_g}$ is independent of the secret. For any $v \in \mathsf{supp}(\mathsf{Share}'(x)_{W_g})$, $\mathsf{Share}'(x)_{W_s}|_{\mathsf{Share}'(x)_{W_g}=v}$ is a convex combination of $\mathsf{Share}'(x)_{W_s}|_{R=r}$ for some different r such that $\mathsf{Share}'(x)_{W_g} = v$ happens.

We claim that $\mathsf{Share}'(x)_{W_s}|_{R=r}$ is a deterministic function of at most k entries of each $y_{s,i}$ for $i \in [\bar{n}]$ and some extra uniform random bits. This is because, as E happens, for those $i \in [n_s]$ with $|\pi(S_i) \cap W| \leq k_C$, the shares in $\pi(S_i) \cap W$ are independent of the secret by the privacy of $(\mathsf{Share}_C, \mathsf{Rec}_C)$. Note that they are also independent of other shares since the construction uses independent randomness for sharing $\hat{y}_{g,i}, i \in [n_g]$ and $\hat{y}_{s,i}, i \in [n_s]$. For those $i \in [n_s]$ with $|\pi(S_i) \cap W| > k_C$, the total number of them is at most $\frac{k}{m_C}$. So $\mathsf{Share}'(x)_{W_s}|_{R=r}$ is computed from at most $\frac{k}{m_C} \times m_C = k$ entries of each $y_{s,i}$ for $i \in [\bar{n}]$ and some extra uniform random bits.

Since the privacy error of $(\mathsf{Share}, \mathsf{Rec})$ is ϵ and $y_{s,i}|_{R=r}, i \in [\bar{n}]$ are computed

using independent uniform random bits, for any $r, r' \in (\Sigma_0^m)^{\bar{n}}$,

$$\mathsf{SD}(\mathsf{Share}'(x)_{W_s}|_{R=r}, \mathsf{Share}'(\sigma)_{W_s}|_{R=r'}) \leq \bar{n}\epsilon.$$

So

$$\mathsf{SD}(\mathsf{Share}'(x)_{W_s}|_{\mathsf{Share}'(x)_{W_g}=v}, \mathsf{Share}'(\sigma)_{W_s}|_{\mathsf{Share}'(\sigma)_{W_g}=v}) \leq \bar{n}\epsilon.$$

As a result,

$$\mathsf{SD}(\mathsf{Share}'(x), \mathsf{Share}'(\sigma)) \leq \bar{n}\epsilon.$$

Thus with probability at least $1 - \bar{n}e^{-\Omega(k/\log n)} - e^{-\Omega(l/\log n)}$ over the fixing of π, $\mathsf{Share}'(x)_W$ and $\mathsf{Share}'(\sigma)_W$ have statistical distance at most $\bar{n}\epsilon$, which means that

$$\mathsf{SD}(\mathsf{Share}'(x), \mathsf{Share}'(\sigma)_W) \leq \bar{n}(\epsilon + e^{-\Omega(k/\log n)}) + e^{-\Omega(l/\log n)}.$$

□

Lemma 2.5.5. *If the non-adaptive privacy error of* $(\mathsf{Share}, \mathsf{Rec})$ *is* ϵ*, then the adaptive privacy error of* $(\mathsf{Share}', \mathsf{Rec}')$ *is* $\bar{n}(\epsilon + e^{-\Omega(k/\log n)}) + e^{-\Omega(l/\log n)}$.

Proof. It follows immediately from Lemma 2.3.10 and 2.5.4.

□

Theorem 2.5.6. *For any* $\epsilon, \eta \in [0,1]$*, any* $n, m \in \mathbb{N}, m \leq n$ *and any constant* $a > 1, \gamma \in (0,1]$*, if there exists an explicit* (n,k) *secret sharing scheme in* $\mathcal{AC}^0$ *with share alphabet* Σ*, message alphabet* Σ_0*, message length* m*, non-adaptive privacy error* ϵ *and reconstruction error* η*, then there exists an explicit* $(n' = O(n^a), \Omega(n'))$ *secret sharing scheme in* $\mathcal{AC}^0$ *with share alphabet* $\Sigma \times [n']$*, message alphabet* Σ *message length* $\Omega(n')$*, adaptive privacy error* $O(n^{a-1}(\epsilon + e^{-\Omega(k/\log n)}) + e^{-\Omega(l/\log n)})$ *and reconstruction error* $O(n^{a-1}\eta)$ *where* $l = \Omega(\frac{mn^{a-1}\log|\Sigma_0|}{\log|\Sigma|})^{1-\gamma}$.

Proof. It follows from Construction 2.5.1, Lemma 2.5.2, 2.5.3, 2.5.5.

□

In step 5 of Construction 2.5.1, if we instead using xor based secret sharing scheme (Theorem 2.2.3) then we can get a even larger privacy threshold, but shorter message length. The proof is similar.

Theorem 2.5.7. *For any $n, m \in \mathbb{N}, m \leq n$, any $\epsilon, \eta \in [0,1]$ and any constant $a > 1, \gamma \in (0,1]$, if there exists an explicit (n,k) secret sharing scheme in $\mathcal{AC}^0$ with share alphabet Σ, message alphabet Σ_0, message length m, non-adaptive privacy error ϵ and reconstruction error η, then there exists an explicit $(n' = O(n^a \log n), (1-\alpha)n')$ secret sharing scheme in $\mathcal{AC}^0$ with share alphabet $\Sigma \times [n']$, message alphabet Σ, message length $\Omega(n^a)$, adaptive privacy error $O(n^{a-1}(\epsilon + 2^{-\Omega(k)}) + 2^{-\Omega(l)})$ and reconstruction error $n^{a-1}\eta$, where $l = \Omega(\frac{mn^{a-1}\log|\Sigma_0|}{\log|\Sigma|})^{1-\gamma}$.*

2.5.2 Binary alphabet

Our construction can be modified to have binary alphabet while keeping the message rate and privacy rate to be constant. We again use the tiny secret sharing schemes from asymptotically good codes as in Section 2.3.

Construction 2.5.8. *For any $n, k, m \in \mathbb{N}$ with $k, m \leq n$, let $(\mathsf{Share}, \mathsf{Rec})$ be an (n,k) secret sharing scheme with alphabet $\{0,1\}$, message length m.*

Let $(\mathsf{Share}_C, \mathsf{Rec}_C)$ be an (n_C, k_C) secret sharing scheme from Lemma 2.3.13 with alphabet $\{0,1\}^{p=O(\log n)}$, message length m_C, where $m_C = \delta_0 n_C$, $k_C = \delta_1 n_C$, $n_C = O(\log n)$ for some constants δ_0, δ_1.

Let $(\mathsf{Share}^*_C, \mathsf{Rec}^*_C)$ be an (n^*_C, k^*_C) secret sharing scheme from Lemma 2.3.13

with alphabet $\{0,1\}$, *message length large enough* m^*_C, *where* $m^*_C = \delta_0 n^*_C = p + O(\log n)$, $n^*_C = \delta_1 n^*_C$.

For any constant $a > 1$, $\gamma > 0$, *we can construct the following* $(n' = O(n^a), k' = \Omega(n')$ *secret sharing scheme* $(\mathsf{Share}', \mathsf{Rec}')$ *with alphabet* $\{0,1\}$, *message length* $m' = \Omega(n')$.

The function $\mathsf{Share}' : \{0,1\}^{m'} \to \{0,1\}^{n'}$ *is as follows.*

1. *Let* $\bar{n} = \Theta(n^{a-1})$ *where the constant factor is large enough.*

2. *Let* $g_\Gamma : \{0,1\}^{m\bar{n}} \to \{0,1\}^{m'}$ *be the* l*-wise independent generator by Theorem 2.4.2, where* $l = \Omega(mn^{a-1})^{1-\gamma}$.

3. *For secret* $x \in \{0,1\}^{m'}$, *we draw a string* $r = (r_1, \ldots, r_{\bar{n}})$ *uniform randomly from* $(\{0,1\}^m)^{\bar{n}}$.

4. *Let* $y = (y_s, y_g)$, *where* $y_s = (y_{s,1}, \ldots, y_{s,\bar{n}}) = (\mathsf{Share}(r_1), \ldots, \mathsf{Share}(r_{\bar{n}})) \in (\{0,1\}^n)^{\bar{n}}$ *and* $y_g = (y_{g,1}, \ldots, y_{g,m'}) = g_\Gamma(r) \oplus x \in \{0,1\}^{m'}$.

5. *Compute* $\hat{y}_s \in ((\{0,1\}^p)^{m_C})^{n_s}$ *from* y_s *by parsing* $y_{s,i}$ *to be blocks over* $(\{0,1\}^p)^{m_C}$ *for every* $i \in [\bar{n}]$, *where* $n_s = \lceil \frac{n}{pm_C} \rceil \bar{n}$.

6. *Compute* $\hat{y}_g \in ((\{0,1\}^p)^{m_C})^{n_g}$ *from* y_g *by parsing* y_g *to be blocks over* $(\{0,1\}^p)^{m_C}$, *where* $n_g = \lceil \frac{m'}{pm_C} \rceil$.

7. *Let*

$$y' = (\mathsf{Share}_C(\hat{y}_{s,1}), \ldots, \mathsf{Share}_C(\hat{y}_{s,n_s}), \mathsf{Share}_C(\hat{y}_{g,1}), \ldots, \mathsf{Share}_C(\hat{y}_{g,n_g})).$$

Parse y' *as* $(y'_1, \ldots, y'_{n^*}) \in (\{0,1\}^p)^{n^*}$, *where* $n^* = (n_s + n_g)n_C$.

8. *Generate a random permutation* $\pi : [n^*] \to [n^*]$ *and compute*

$$z(x) = (\mathsf{Share}^*_C(y'_{\pi^{-1}(1)} \circ \pi^{-1}(1)), \ldots, \mathsf{Share}^*_C(y'_{\pi^{-1}(n^*)} \circ \pi^{-1}(n^*))) \in (\{0,1\}^{n^*_C})^{n^*}.$$

9. *Parse* $z(x)$ *to be bits and output.*

The function $\mathsf{Rec}' : \{0,1\}^{n'} \to \{0,1\}^{m'}$ *is as follows.*

1. *Parse the input bits to be* $z = (z_1, \ldots, z_{n^*}) \in (\{0,1\}^{n^*_C})^{n^*}$.
2. *For every* $i \in [n^*]$, *let* $(y'_{\pi^{-1}(i)} \circ \pi^{-1}(i)) = \mathsf{Rec}^*_C(z_i)$ *to get* y'.
3. *Parse* $y' = (y'_s, y'_g)$ *where* $y'_s = (y'_{s,1}, \ldots, y'_{s,n_s}) \in (\{0,1\}^{pn_C})^{n_s}$, $y'_g = (y'_{g,1}, \ldots, y'_{g,n_g}) \in (\{0,1\}^{pn_C})^{n_g}$.
4. *Let*

$$\hat{y}_s = (\mathsf{Rec}_C(y'_{s,1}), \ldots, \mathsf{Rec}_C(y'_{s,n_s})), \hat{y}_g = (\mathsf{Rec}_C(y'_{g,1}), \ldots, \mathsf{Rec}_C(y'_{g,n_g})).$$

5. *Parse* $\hat{y}_s$ *to get* y_s.
6. *Parse* $\hat{y}_g$ *to get* y_g
7. *Let* $r = (\mathsf{Rec}(y_{s,1}), \ldots, \mathsf{Rec}(y_{s,\bar{n}}))$.
8. *Output*

$$\mathsf{Rec}'(z) = g_\Gamma(r) \oplus y_g.$$

Lemma 2.5.9. *If* $(\mathsf{Share}, \mathsf{Rec})$ *can be computed by* $\mathcal{AC}^0$ *circuits, then* $(\mathsf{Share}', \mathsf{Rec}')$ *can be computed by* $\mathcal{AC}^0$ *circuits.*

Proof Sketch. The construction is similar to that of Construction 2.5.1. As $n_C = O(\log n), n'_C = O(\log n)$, $n^*_C = O(\log n)$, we know that $(\mathsf{Share}_C, \mathsf{Rec}_C)$ and $\mathsf{Share}^*_C, \mathsf{Rec}^*_C$ can be computed by $\mathcal{AC}^0$ circuits by Lemma 2.2.9 part 2.

So the overall construction can be computed by $\mathcal{AC}^0$ circuits.

□

Lemma 2.5.10. *If the reconstruction error of* $(\mathsf{Share}, \mathsf{Rec})$ *is* η*, then the reconstruction error of* $(\mathsf{Share}', \mathsf{Rec}')$ *is* $\eta' = \bar{n}\eta$.

Proof. As $(\mathsf{Share}_C, \mathsf{Rec}_C)$ and $(\mathsf{Share}^*_C, \mathsf{Rec}^*_C)$ have perfect reconstructions, the error only occurs when we apply Rec. So we can compute each $r_i, i = 1, \ldots, \bar{n}$ correctly except with error η. By the union bound, with probability $1 - \bar{n}\eta$, r is correctly computed. Once we can compute r correctly, the secret $x = g_\Gamma(r) \oplus y_g$. It remains to show the correctness of y_g. From z we can get the value of every entry of y', since for every $i \in [n^*]$, $(y'_{\pi^{-1}(i)} \circ \pi^{-1}(i)) = \mathsf{Rec}^*_C(z_i)$ and we can get y' by putting each value into its correct position. Thus y'_g is correct as it is part of y'. By noting that Rec_C has no reconstruction error, $\hat{y}_g$ is correct. As y_g is from $\hat{y}_g$ by parsing, it is also correct.

□

Lemma 2.5.11. *If the non-adaptive privacy error of* $(\mathsf{Share}, \mathsf{Rec})$ *is* ϵ*, then the non-adaptive privacy error of* $(\mathsf{Share}', \mathsf{Rec}')$ *is* $\bar{n}(\epsilon + e^{-\Omega(k/\log^2 n)}) + e^{-\Omega(l/\log^2 n)}$.

Proof Sketch. Let $k' = 0.9\delta_1^2 n'$. We need to show that there exists a distribution $\mathcal{D}$ such that for any $W \subseteq [n']$ with $|W| \leq k'$, for every $x \in \{0,1\}^{m'}$,

$$\mathsf{SD}(\mathsf{Share}'(x)_W, \mathcal{D}) \leq \bar{n}(\epsilon + e^{-\Omega(k/\log^2 n)}) + e^{-\Omega(l/\log^2 n)}.$$

Let $\mathcal{D}$ be $\mathsf{Share}'(\sigma)_W$ for an arbitrary $\sigma \in \{0,1\}^{m'}$.

Consider an arbitrary observation $W \subseteq [n']$ with $|W| \leq k'$. For at least $1 - 0.9\delta_1$ fraction of the blocks $z_i, i = 1, \ldots, n^*$, at most δ_1 fraction of the bits in each block can be observed, because otherwise the number of observed shares is more than $0.9\delta_1 \times \delta_1 n' = 0.9\delta_1^2 n'$. Let W^* be the index sequence of those blocks which have more than δ_1 fraction of their bits observed. Let $|W^*| = k^*$ which is at most $0.9\delta_1 n^*$.

Consider every $i \notin W^*$. The distribution of z_i is independent of $y'_{\pi^{-1}(i)} \circ \pi^{-1}(i)$ by the privacy of $(\mathsf{Share}^*_C, \mathsf{Rec}^*_C)$. Since every Share_0 function uses independent randomness, z_i is also independent of $z_{i'}$ for every $i' \in [n^*]$ with $i' \neq i$. So we only have to show that

$$\mathsf{SD}(z_{W^*}(x), z_{W^*}(\sigma)) \leq \bar{n}(\epsilon + e^{-\Omega(k/\log^2 n)}) + e^{-\Omega(l/\log^2 n)}.$$

For every $i \in [n_s]$, let $S_i = \{n_C(i-1)+1, \ldots, n_C i\}$ and X_i be the boolean random variable such that $X_i = 1$ is the event $|\pi(S_i) \cap W^*| > k_C$. Let $\mathcal{S} = \{S_1, \ldots, S_{n_s}\}$. Note that $\forall i \in [n_s], \mathbb{E}[|\pi(S_i) \cap W^*|] \leq 0.9\delta_1 n_C = 0.9k_C$.

For every $i \in [\bar{n}]$, let E_i be the event that $\sum_{j=(i-1)\lceil \frac{n}{pm_C} \rceil + 1}^{i\lceil \frac{n}{pm_C} \rceil} X_j \leq \frac{k}{pm_C}$. We take $\bar{n}$ to be large enough such that $n_C \lceil \frac{n}{pm_C} \rceil \leq \frac{0.9 \times 0.1}{1 + 0.9 \times 0.1} |W^*|$. By Lemma 2.3.8, for every $i \in [\bar{n}]$.

$$1 - \Pr[E_i] \leq e^{-2k/(pm_C) + (e^2-1)e^{-\Omega(0.9\delta_1^2 n_C)} \lceil \frac{n}{pm_C} \rceil}.$$

We take $n_C = O(\log n)$ to be large enough such that the probability is at most $e^{-\Omega(k/(pm_C))} \leq e^{-\Omega(k/\log^2 n)}$.

For every $i \in [n_g]$, $T_i = \{n'_C(i-1)+1, \ldots, n_C i\}$ and Y_i be the event that $|\pi(T_i) \cap W^*| > k_C$. Let $\mathcal{T} = \{T_1, \ldots, T_{n_g}\}$. Let $Y = \sum_{i \in [n_g]} Y_i$. Note that $\mathbb{E}[|\pi(T_i) \cap W^*|] \leq 0.9\delta_1 n_C = 0.9k_C$. Let E_g be the event such that $Y \leq \frac{l}{pm_C}$. Since

$\bar{n}$ is large enough, we have $n_C n_g \leq \frac{0.9\times 0.1}{1+0.9\times 0.1}|W^*|$. By Lemma 2.3.8,

$$1 - \Pr[E_g] \leq e^{-2l/(pm_C)+(e^2-1)e^{-\Omega(0.9\delta_1^2 n_C)}n_g}.$$

We take $n_C = O(\log n)$ to be large enough such that the probability is at most $e^{-\Omega(l/(pm_C))} \leq e^{-\Omega(l/\log^2 n)}$.

Let E be the event that $(\bigcap_{i\in[\bar{n}]} E_i) \cap E_g$. By the union bound, $\Pr[E] \geq 1 - \bar{n}e^{-\Omega(k/(\log^2 n))} - e^{-\Omega(l/(\log^2 n))}$. We claim that $z_{W^*}(x)|E$ and $z_{W^*}(\sigma)|E$ have statistical distance at most $\bar{n}\epsilon$. The reason is as follows.

We fix a permutation π for which E happens. Let $W_s = (\bigcup_{i\in[n_s]} S_i) \cap W^*$, $W_g = (\bigcup_{i\in[n_g]} T_i) \cap W^*$. Let R be the random variable which corresponds to the random choice of r.

We claim that $z_{W_g}(x)$ is a deterministic function of at most l entries of y_g and some extra uniform random bits. As E_g happens, for those $i \in [n_g]$ with $|\pi(T_i) \cap W^*| \leq k_C$, the blocks indexed by $\pi(T_i) \cap W^*$ are independent of the secret by the privacy of $(\mathsf{Share}_C, \mathsf{Rec}_C)$. Note that they are also independent of other blocks since the construction uses independent randomness for sharing $\hat{y}_{g,i}, i \in [n_g]$ and $\hat{y}_{s,i}, i \in [n_s]$. For those $i \in [n_g]$ with $|\pi(T_i) \cap W^*| > k_C$, the total number of them is at most $\frac{l}{pm_C}$. So $z_{W_g}(x)$ is computed from at most $\frac{l}{pm_C} \times pm_C = l$ entries of y_g and some extra uniform random bits.

As $g_\Gamma(\cdot)$ is an l-wise independent generator, the distribution of $z_{W_g}(x)$ is independent of the secret. For any $v \in \mathsf{supp}(z_{W_g}(x))$, consider $z_{W_s}(x)|_{z_{W_g}(x)=v}$ and $z_{W_s}(\sigma)|_{z_{W_g}(\sigma)=v}$. Note that $z_{W_s}(x)|_{z_{W_g}(x)=v}$ is a convex combination of $z_{W_s}(x)|_{R=r}$ for some different r such that $z_{W_g}(x) = v$ happens.

We claim that $z_{W_s}(x)|_{R=r}$ is a deterministic function of at most k entries of

each $y_{s,i}$ for $i \in [\bar{n}]$ and some extra uniform random bits. This is because, as E happens, for those $i \in [n_s]$ with $|\pi(S_i) \cap W^*| \leq k_C$, the shares in $\pi(S_i) \cap W^*$ are independent of the secret by the privacy of $(\mathsf{Share}_C, \mathsf{Rec}_C)$. Note that they are also independent of other shares since the construction uses independent randomness for sharing $\hat{y}_{g,i}, i \in [n_g]$ and $\hat{y}_{s,i}, i \in [n_s]$. For those $i \in [n_s]$ with $|\pi(S_i) \cap W^*| > k_C$, the total number of them is at most $\frac{k}{pm_C}$. So $z_{W_s}(x)|_{R=r}$ is computed from at most $\frac{k}{pm_C} \times pm_C = k$ entries of each $y_{s,i}$ for $i \in [\bar{n}]$ and some extra uniform random bits.

Since the privacy error of $(\mathsf{Share}, \mathsf{Rec})$ is ϵ and every Share function uses independent uniform random bits, for any $r, r' \in (\Sigma_0^m)^{\bar{n}}$,

$$\mathsf{SD}(z_{W_s}(x)|_{R=r}, z_{W_s}(\sigma)|_{R=r}) \leq \bar{n}\epsilon.$$

So

$$\mathsf{SD}(z_{W_s}(x)|_{z_{W_g}(x)=v}, z_{W_s}(\sigma)|_{z_{W_g}(\sigma)=v}) \leq \bar{n}\epsilon.$$

As a result,

$$\mathsf{SD}(z_{W_s}(x), z_{W_s}(\sigma)) \leq \bar{n}\epsilon.$$

Thus with probability at least $1 - \bar{n}e^{-\Omega(k/(\log^2 n))} - e^{-\Omega(l/(\log^2 n))}$ over the fixing of π, $z_{W^*}(x)$ and $z_{W^*}(\sigma)$ have statistical distance at most $\bar{n}\epsilon$, which means that

$$\mathsf{SD}(z_{W^*}(x), z_{W^*}(\sigma)) \leq \bar{n}(\epsilon + e^{-\Omega(k/\log^2 n)}) + e^{-\Omega(l/\log^2 n)}.$$

□

Using Lemma 2.5.11 and a similar argument as in Lemma 2.3.19, we can get adaptive privacy as follows.

Lemma 2.5.12. *If the non-adaptive privacy error of* $(\mathsf{Share}, \mathsf{Rec})$ *is* ϵ*, then the adaptive privacy error of* $(\mathsf{Share}', \mathsf{Rec}')$ *is* $\bar{n}(\epsilon + e^{-\Omega(k/\log^2 n)}) + e^{-\Omega(l/\log^2 n)}$.

Proof. Let W be the adaptive observation of length k'. Let $W' = [\lceil k'/k_C^* \rceil]$. By Lemma 2.3.18, there exists a deterministic function f such that for $x, x' \in \{0,1\}^{m'}$, $\mathsf{SD}(\mathsf{Share}'(x)_W, \mathsf{Share}(x')_W) = \mathsf{SD}(f(\mathsf{Share}'(x)_{W'} \circ R \circ S), f(\mathsf{Share}'(x')_{W'} \circ R \circ S))$ where R, S are as defined in Lemma 2.3.18 which are independent of $\mathsf{Share}'(x)_{W'}$ and $\mathsf{Share}'(x')_{W'}$. By Lemma 2.2.2, we know that

$$\mathsf{SD}(f(\mathsf{Share}'(x)_{W'} \circ R \circ S), f(\mathsf{Share}'(x')_{W'} \circ R \circ S)) \leq \mathsf{SD}(\mathsf{Share}'(x)_{W'}, \mathsf{Share}'(x')_{W'}).$$

By Lemma 2.5.11 we know that

$$\mathsf{SD}(\mathsf{Share}'(x)_{W'}, \mathsf{Share}'(x')_{W'}) \leq \bar{n}(\epsilon + e^{-\Omega(k/\log^2 n)}) + e^{-\Omega(l/\log^2 n)}.$$

So

$$\mathsf{SD}(\mathsf{Share}'(x)_W, \mathsf{Share}'(x')_W) \leq \bar{n}(\epsilon + e^{-\Omega(k/\log^2 n)}) + e^{-\Omega(l/\log^2 n)}.$$

□

Theorem 2.5.13. *For any* $\epsilon, \eta \in [0,1]$*, any* $n, m \in \mathbb{N}, m \leq n$ *and any constant* $a > 1, \gamma > 0$*, if there exists an explicit* (n,k) *secret sharing scheme in* $\mathcal{AC}^0$ *with alphabet* $\{0,1\}$*, message length* m*, non-adaptive privacy error* ϵ *and reconstruction error* η*, then there exists an explicit* $(n' = O(n^a), \Omega(n'))$ *secret sharing scheme in* $\mathcal{AC}^0$ *with alphabet* $\{0,1\}$*, message length* $\Omega(n')$*, adaptive privacy error* $O(n^{a-1}(\epsilon + 2^{-\Omega(k/\log^2 n)}) + 2^{-\Omega((mn^{a-1})^{1-\gamma}/\log^2 n)})$ *and reconstruction error* $O(n^{a-1}\eta)$.

Proof. It follows from Construction 2.5.8, Lemma 2.5.9, 2.5.10, 2.5.12.

□

2.6 Instantiation

The Minsky-Papert function [MP88] gives a secret sharing scheme in $\mathcal{AC}^0$ with perfect privacy.

Theorem 2.6.1 ([MP88])**.** *For any $n \in \mathbb{N}$, there exists an explicit $(n, n^{\frac{1}{3}})$ secret sharing scheme in $\mathcal{AC}^0$ with alphabet $\{0,1\}$, message length 1, perfect privacy and reconstruction.*

Combining our techniques with Theorem 2.6.1, we have the following results.

Theorem 2.6.2. *For any $n \in \mathbb{N}$, any constant $\alpha \in (0,1], \beta \in [0,1)$, there exists an explicit $(n, (1-\alpha)n)$ secret sharing scheme in $\mathcal{AC}^0$ with share alphabet $\{0,1\}^{O(\log n)}$, message alphabet $\{0,1\}$, message length $m = n^{\beta}$, adaptive privacy error $2^{-\Omega((\frac{n}{m \log n})^{1/3})}$ and perfect reconstruction.*

Proof. It follows from Theorem 2.6.1 and 2.3.12. We use the $(n_0, n_0^{1/3})$ secret sharing scheme from Theorem 2.6.1 to instantiate Theorem 2.3.12. So $n = O(n_0^a \log n_0)$ for some constant $a > 1$. The message length is $O(n/(n_0 \log n))$. Since $n_0 = \frac{n}{m \log n}$, the privacy error is $2^{-\Omega((\frac{n}{m \log n})^{1/3})}$. The share alphabet is $\{0,1\} \times [n]$. □

Note that when $\beta = 0$, this is a scheme sharing 1 bit. Next we give our theorem for secret sharing schemes with binary alphabet, constant secret rate and constant privacy rate.

Theorem 2.6.3. *For any $n \in \mathbb{N}$, for any constant $\gamma \in (0, 1/4)$, there exists an explicit $(n, \Omega(n))$ secret sharing scheme in $\mathcal{AC}^0$ with alphabet $\{0,1\}$, message*

length $m = \Omega(n)$*, adaptive privacy error* $2^{-\Omega(n^{\frac{1}{4}-\gamma})}$ *and perfect reconstruction.*

Proof. It follows from Theorem 2.6.1 and 2.5.13.

Let $(n_0, n_0^{1/3})$ be the secret sharing scheme of Theorem 2.6.1 with message length $m_0 = 1$. Let $n = O(n_0^a)$ for some constant $a > 1$. For any constant $\beta \in (0,1)$, let $n_0^{1/3} = (m_0 n_0^{a-1})^{1-\beta}$. Then $a = \frac{4-3\beta}{3(1-\beta)}$. So $n_0 = O(n^{\frac{3(1-\beta)}{4-3\beta}})$. Hence by Theorem 2.5.13, we have the desired secret sharing scheme with the privacy error $2^{-\Omega(n^{\frac{1-\beta}{4-3\beta}}/\log^2 n)}$.

□

2.7 Extensions and Applications

2.7.1 Robust secret sharing

Our secret sharing schemes can be made robust by using robust secret sharing schemes and authentication techniques in small blocks.

We will use cyclic shifting of indices in some constructions in this section. For any index i in some index set S, for any $j \in \mathbb{N}$, $i \gg j$ is the index obtained from i by doing right cyclic shifting for j positions, whereas $i \ll j$ is the index obtained from i by doing left cyclic shifting for j positions.

Theorem 2.7.1 ([Che16])**.** *For any* $n \in \mathbb{N}$*, any constant* $\rho < 1/2$*, there exists an* $(n, \Omega(n))$ *robust secret sharing scheme, with alphabet* $\{0,1\}^{O(1)}$*, message length* $\Omega(n)$*, perfect privacy, robustness parameter* $d = \rho n$ *and reconstruction error* $2^{-\Omega(n)}$.

Also we need the following theorem about computing approximate majorities.

Theorem 2.7.2 ([Vio09]). *For every $n \in \mathbb{N}$, there exists an explicit depth 3 circuit $C_n : \{0,1\}^n \to \{0,1\}$ which decides whether the fraction of 1's in the input is at least $2/3$.*

We use concatenations of the schemes from Theorem 2.7.1 to get the following robust secret sharing scheme in $\mathcal{AC}^0$ with poly-logarithmic number of shares.

Lemma 2.7.3. *For any $n \in \mathbb{N}$, any constant $a \in \mathbb{N}$, any $\epsilon = 1/\mathsf{poly}(n)$, there exists an $(n_0 = O(\log^a n), k_0 = \Omega(n_0))$ robust secret sharing scheme in $\mathcal{AC}^0$ (in n), with share alphabet $\{0,1\}^{O(1)}$, message alphabet $\{0,1\}$, message length $\Omega(n_0)$, perfect privacy, robustness parameter $\Omega(n_0)$, reconstruction error ϵ.*

Proof. Let $(\mathsf{RShare}_1, \mathsf{RRec}_1)$ be an $(n_1 = O(\log n), k_1 = \Omega(n_1))$ robust secret sharing scheme from Theorem 2.7.1 with message length m_1, robustness parameter d_1, share alphabet $\{0,1\}^{p=O(1)}$.

We use induction on a.

For $a = 1$, as the output length is $O(\log n)$, by Lemma 2.2.9, the sharing and reconstruction can both be done in $\mathcal{AC}^0$. Other properties follow from Theorem 2.7.1.

Assume the conclusion holds for some $a \geq 1$. So there exists an $(n_a = O(\log^a n), k_a = \Omega(n_a))$ robust secret sharing scheme meeting the requirements, with message length m_a, message alphabet $\{0,1\}$, robustness parameter d_a, share alphabet $\{0,1\}^p$. Consider $a+1$. For any $x \in \{0,1\}^{O(\log^{a+1} n)}$, let RShare_{a+1} be constructed as follows. Split x into blocks $(\bar{x}_1, \ldots, \bar{x}_{m_1}) \in (\{0,1\}^{m_a})^{m_1}$. Let $\bar{y}_i = \mathsf{RShare}_1(\bar{x}_{1,i}, \ldots, \bar{x}_{m_1,i})$, $i = 1, \ldots, m_a$, where $\bar{x}_{j,i}$ is the ith bit of $\bar{x}_j$, $j = 1, \ldots, m_1$. Let $\tilde{x}_i = (\bar{y}_{1,i}, \ldots, \bar{y}_{m_a,i})$, $i = 1, \ldots, n_1$, where $\bar{y}_{j,i}$ is the ith bit of $\bar{y}_j$, $j = 1, \ldots, m_a$.

Let $y_i = \mathsf{RShare}_a(\tilde{x}_i)$, $i = 1, \ldots, n_1$. Let $\mathsf{RShare}_{a+1}(x) = (y_1, \ldots, y_{n_1})$. The message length is $m_{a+1} = m_1 \cdot m_a$. The privacy is $k_{a+1} = k_1 \cdot k_a = \Omega(n_a)$. This is because, if the adversary can see at most $k_1 \cdot k_a$ shares, then at most k_1 of $\tilde{x}_1, \ldots, \tilde{x}_{n_1}$ are observed. So by the privacy of Share_1, $\bar{x}_1, \ldots, \bar{x}_{m_1}$ are not observed. Thus x is not observed. The robustness is $d_{a+1} = d_1 d_a = \Omega(n_a)$, due to a similar argument as for the privacy. RShare_{a+1} can be computed by $\mathcal{AC}^0$ circuits since both RShare_1 and RShare_a can be computed by $\mathcal{AC}^0$ circuits. The reconstruction is in $\mathcal{AC}^0$ since we can first apply RRec_a on y_i for every $i = 1, \ldots, n_1$. By assumption this is in $\mathcal{AC}^0$. Then we apply RRec_1 on $\bar{y}_i$ for every $i = 1, \ldots, m_a$. This is in $\mathcal{AC}^0$ by the base case. The reconstruction error is still $1/\mathsf{poly}(n)$ since both $(\mathsf{RShare}_1, \mathsf{RRec}_1)$ and $(\mathsf{RShare}_a, \mathsf{RRec}_a)$ have reconstruction error $1/\mathsf{poly}(n)$.

□

Next, we give our construction of robust secret sharing scheme with "asymptotically good" parameters.

Theorem 2.7.4. *For any $n \in \mathbb{N}$, any $\eta = \frac{1}{\mathsf{poly}(n)}$, there exists an explicit $(n, \Omega(n))$ robust secret sharing scheme in $\mathcal{AC}^0$ with share alphabet $\{0,1\}^{O(1)}$, message alphabet $\{0,1\}$, message length $m = \Omega(n)$, non-adaptive privacy error $2^{-n^{\Omega(1)}}$, non-adaptive robustness $\Omega(n)$ and reconstruction error η.*

Proof Sketch. We modify Construction 2.5.8. Let δ_0, δ_1, ρ be some proper constants in $(0,1)$.

Let $(\mathsf{RShare}_C, \mathsf{RRec}_C)$ be an (n_C, k_C) secret sharing scheme from Theorem 2.7.3 with share alphabet $\{0,1\}^{p=O(\log^2 n')}$, message alphabet $\{0,1\}$, message length m_C, where $m_C = \delta_0 n_C$, $k_C = \delta_1 n_C$, $n_C = O(\log n')$, robustness parameter ρn_C.

Also let $(\mathsf{RShare}^*_C, \mathsf{RRec}^*_C)$ be an (n^*_C, k^*_C) secret sharing scheme from Theorem 2.7.3 with share alphabet $\Sigma = \{0,1\}^{O(1)}$, message alphabet $\{0,1\}$, message length $m^*_C = p + O(\log^2 n)$, where $m^*_C = \delta_0 n^*_C$, $k^*_C = \delta_1 n^*_C$, robustness parameter ρn^*_C.

The robust secret sharing scheme construction is the same as that of Construction 2.5.1 except the following modifications. We replace $(\mathsf{Share}_C, \mathsf{Rec}_C)$ and $(\mathsf{Share}^*_C, \mathsf{Rec}^*_C)$ by their corresponding robust ones $(\mathsf{RShare}_C, \mathsf{RRec}_C)$ and $(\mathsf{RShare}^*_C, \mathsf{RRec}^*_C)$. For the share function, we replace the last two steps in Construction 2.5.8 by the following.

- Generate a random permutation $\pi : [n^*] \to [n^*]$.
- Randomly pick $l' = O(\log n^*)$ indices $r'_1, \ldots, r'_{l'} \in [n^*]$ and let $r' = (r'_1, \ldots, r'_{l'})$.
- For each block $y'_{\pi^{-1}(i)} \circ \pi^{-1}(i), i = 1, \ldots, n^*$, we attach r' and $\pi^{-1}(i \gg r'_1), \pi^{-1}(i \gg r'_2), \ldots, \pi^{-1}(i \gg r'_{l'})$ to it. That is, the ith block is

$$\tilde{y}_i = y'_{\pi^{-1}(i)} \circ \pi^{-1}(i) \circ \pi^{-1}(i \gg r'_1) \circ \cdots \circ \pi^{-1}(i \gg r'_{l'}) \circ r'.$$

- Compute $z(x) = (\mathsf{RShare}^*_C(\tilde{y}_1), \ldots, \mathsf{RShare}^*_C(\tilde{y}_{n^*}))$.
- Parse z to be shares over $\Sigma^{n'=n^*_C n^*}$ and output.

For the reconstruction function we replace the first two steps with the following.

- Parse the input to be blocks each of length n^*_C and apply RRec^*_C on every block to get $\tilde{y}$.
- Compute r' by taking the approximate majority of the r's in $\tilde{y}_i, i = 1, \ldots, n^*$.
- $\forall i \in [n']$, take the approximate majority of the l backups of $\pi^{-1}(i)$ to reconstruct $\pi^{-1}(i)$.

- Compute y' using the recovered indices and $\tilde{y}$.

We claim that, we get a $(n', \Omega(n'))$ robust secret sharing scheme with share alphabet $\{0,1\}^{O(1)}$, message alphabet $\{0,1\}$, message length $m' = \Omega(n')$, non-adaptive privacy error $2^{-n'^{\Omega(1)}}$, non-adaptive robust parameter $\Omega(n')$ and reconstruction error η.

The non-adaptive privacy can be proved in the same way as that of Lemma 2.5.11. Note that for each $i \in [n^*]$ we attach additional information about the indices. But this gives no more information about the secret since we are considering the non-adaptive case.

What we need to prove is that the reconstruction works under non-adaptive adversaries. Assume that the adversary corrupts at most $\min(0.9\rho^2, \rho/3)$ fraction of shares. Thus for at most $1/3$ fraction of $z_i, i \in [n^*]$, the fraction of corrupted shares is more than ρ. Because otherwise the total fraction of corrupted shares is more than $\rho/3$. Hence for at least $2/3$ fraction of $z_i, i \in [n^*]$, the fraction of corrupted shares is at most ρ. By the robustness of $(\mathsf{RShare}^*_C, \mathsf{RRec}^*_C)$, at least $2/3$ fraction of $\tilde{y}_i, i \in [n^*]$ can be reconstructed correctly. By Theorem 2.7.2, we can reconstruct r' correctly.

For any $i \in [n^*]$, the event $\pi^{-1}(i)$ can be recovered correctly with probability $1 - e^{-\Theta(\log n')} = 1 - \mathsf{poly}(n')$ by a Chernoff bound, since the l' indices $r'_1, \ldots, r'_{l'}$ are independently chosen and at most $\rho/3$ fraction of $\tilde{y}_1, \ldots, \tilde{y}_{n^*}$ cannot be reconstructed correctly. By the union bound with probability $1 - \mathsf{poly}(n')$, $\forall i \in [n^*]$, $\pi^{-1}(i)$ can be recovered correctly. Before applying RRec_C on every y'_i, we bound the number of corrupted bits (including the blanks) for y'_i. The probability that the fraction of corrupted bits in y'_i is at most ρ is at least $1 - 1/\mathsf{poly}(n')$ by Lemma 2.3.7. Once for every block $y'_i, i = 1, \ldots, n^*$, the corrupted rates are at most ρ , we can finally get the

correct secret. By the union bound, we know the probability that this happens is at least $1 - 1/\mathsf{poly}(n')$.

The construction is still in $\mathcal{AC}^0$. We only need to show that the modified parts can be computed in $\mathcal{AC}^0$. For the share function, generating random permutation is by Lemma 2.3.1. Additions of indices can be computed by $\mathcal{AC}^0$ circuits by Lemma 2.2.9 since the indices are recorded by $O(\log n')$ bits. Also note that RShare^*_C is from Lemma 2.7.3. For the reconstruction, note that RRec^*_C is from Lemma 2.7.3. The approximate majority is from Theorem 2.7.2. So they all can be computed by $\mathcal{AC}^0$ circuits. The tests which check the equivalence of indices are in $\mathcal{AC}^0$ by Lemma 2.2.9, as the input length is $O(\log n')$.

We still use Theorem 2.6.1 to instantiate the scheme.

□

2.7.2 Stochastic error correcting code

Using our general strategy, we can also construct stochastic error correcting codes in $\mathcal{AC}^0$ which can resist additive errors ([GS16]).

One important component of our construction is the following “tiny" codes. It is constructed by classic code concatenation techniques.

Lemma 2.7.5. *For any $n \in \mathbb{N}$, any constant $a \in \mathbb{N}$, there exists an asymptotically good binary $(n_0 = O(\log^a n), m_0, d_0)$ code C such that the encoding and decoding can both be computed by $\mathcal{AC}^0$ circuits of size $\mathsf{poly}(n)$.*

Proof. Let C_1 be an $(n_1 = O(\log n), m_1, d_1)$ binary code which is asymptotically good.

We use induction on a.

For $a = 1$, as the code length is $O(\log n)$ and there are plenty of asymptotically good binary codes construction, by Lemma 2.2.9, the encoding and decoding can both be done in $\mathcal{AC}^0$. So our conclusion holds in this case.

Assume the conclusion holds for some $a \geq 1$. So there exists an asymptotically good binary $(n_a = O(\log^a n), m_a, d_a)$ code C_a. Consider $a+1$. For any $x \in \{0,1\}^{O(\log^{a+1} n)}$, let $C_{a+1}(x)$ be computed as the following codes concatenation. Parse x into blocks of length m_a, which is $(\bar{x}_1, \ldots, \bar{x}_{m_1})$. Let $\bar{y}_i = C_1(\bar{x}_{1,i}, \ldots, \bar{x}_{m_1,i})$, $i = 1, \ldots, m_a$, where $\bar{x}_{j,i}$ is the ith bit of $\bar{x}_j$. Let $\tilde{x}_i = (\bar{y}_{1,i}, \ldots, \bar{y}_{m_a,i})$, $i = 1, \ldots, n_1$, where $\bar{y}_{j,i}$ is the ith bit of $\bar{y}_j$, $j = 1, \ldots, m_a$. Let $y_i = C_a(\tilde{x}_i)$, $i = 1, \ldots, n_1$. Let $C_{a+1}(x) = (y_1, \ldots, y_{n_1})$. The message length is $m_{a+1} = m_1 \cdot m_a$. The distance is $d_{a+1} = d_1 \cdot d_a$. So C_{a+1} is still an asymptotically good code due to that a is a constant. The encoding can be computed by $\mathcal{AC}^0$ circuits since the encoding of both C_1 and C_a can be computed by $\mathcal{AC}^0$ circuits. The decoding is in $\mathcal{AC}^0$ since we can first decode y_i for every $i = 1, \ldots, n_1$. By assumption this is in $\mathcal{AC}^0$. Then we decode $\bar{y}_i$ for every $i = 1, \ldots, m_a$. This is in $\mathcal{AC}^0$ by the base case.

□

Here we give the construction of stochastic error correcting codes in $\mathcal{AC}^0$ which are "asymptotically good".

Construction 2.7.6. *For any $n \in \mathbb{N}$, we construct the following $(n, m = \Omega(n), \rho = \Omega(1))$ stochastic error correcting code.*

Let δ_0, δ_1 be some proper constants in $(0,1)$.

Let $(\mathsf{Enc}_0, \mathsf{Dec}_0)$ be an asymptotically good (n_0, m_0, d_0) error correcting code

with alphabet $\{0,1\}^p$, $n_0 = O(\log n)$, $m_0 = \delta_0 n_0$, $d_0 = \delta_1 n_0$. *In fact we can realize this code by applying an asymptotically good binary code, having the same rate, in parallel* p *times.*

Let $(\mathsf{Enc}_1, \mathsf{Dec}_1)$ *be an asymptotically good* (n_1, m_1, d_1) *error correcting code from Lemma 2.7.5 with alphabet* $\{0,1\}$, $n_1 = p + O(\log n)$, $m_1 = \delta_0 n_1 = O(p)$, $d_1 = \delta_1 n_0$.

Encoding function $\mathsf{Enc} : \{0,1\}^{m=\Omega(n)} \to \{0,1\}^n$ *is a random function which is as follows.*

1. *On input* $x \in \{0,1\}^m$, *split* x *into blocks of length* pm_0 *such that* $x = (\bar{x}_1, \ldots, \bar{x}_{m/(pm_0)}) \in (\{0,1\}^{pm_0})^{m/(pm_0)}$.
2. *Compute* $(\mathsf{Enc}_0(\bar{x}_1), \ldots, \mathsf{Enc}_0(\bar{x}_{m/(pm_0)}))$ *and parse it to be* $y = (y_1, \ldots, y_{n'}) \in (\{0,1\}^p)^{n'}, n' = m/(\delta_0 p)$.
3. *Generate a random permutation* $\pi : [n'] \to [n']$.
4. *Randomly pick* $l = O(\log n)$ *different indices* $r_1, \ldots, r_l \in [n']$ *and let* $r = (r_1, \ldots, r_l)$.
5. *For every* $i \in [n']$, *let* $\tilde{y}_i = (y_{\pi^{-1}(i)}, \pi^{-1}(i), \pi^{-1}(i \gg r_1), \ldots, \pi^{-1}(i \gg r_l), r)$.
6. *Output* $z = (\mathsf{Enc}_1(\tilde{y}_1), \ldots, \mathsf{Enc}_1(\tilde{y}_{n'})) \in (\{0,1\}^{n_1})^{n'}$.

Decoding function $\mathsf{Dec} : \{0,1\}^{n=n_1 n'} \to \{0,1\}^m$ *is as follows.*

1. *On the input* z, *apply* Dec_1 *on every block of length* n_0 *to get* $\tilde{y}$.

2. *Take the majority of the r in every $\tilde{y}_i, i \in [n']$ to get r.*

3. *$\forall i \in [n']$, take the approximate majority of the l backups of $\pi^{-1}(i)$ to reconstruct $\pi^{-1}(i)$.*

4. *Compute the entries of y using the recovered indices and $\tilde{y}$.*

5. *Apply Dec_0 on every block of y of length pn_0 to get x.*

Theorem 2.7.7. *For any $n \in \mathbb{N}$, any $\epsilon = 1/\mathsf{poly}(n)$, there exists an explicit $(n, m = \Omega(n), \rho = \Omega(1))$ stochastic binary error correcting code with decoding error ϵ, which can be computed by $\mathcal{AC}^0$ circuits.*

The proof here is similar to the proof of Theorem 2.7.4.

Proof Sketch. We claim that Construction 2.7.6 gives such a code.

Let $\rho = 0.9(\delta_1/3)^2$. So for at most $0.9(\delta_1/3)$ fraction of blocks $z_1, \ldots, z'_n$, each of them has at least $(\delta_1/3)$ fraction of bits being corrupted, since otherwise the overall corrupted bits is larger than $0.9(\delta_1/3) \times (\delta_1/3)n_1n' = \rho n$. For blocks of $z_1, \ldots, z_n$ with less than $(\delta_1/3)$ fraction of bits being corrupted, they can be decoded correctly since $d_1/2 = \delta_1 n_1/2$. Thus we know that r can be reconstructed correctly through taking approximate majorities.

For any $i \in [n']$, as $r_1, \ldots, r_l$ are randomly chosen and at most $0.9(\delta_1/3)$ fraction of $\tilde{y}_1, \ldots, \tilde{y}_{n'}$ can not be computed correctly, by a Chernoff bound, the probability that $\pi^{-1}(i)$ is correctly computed can be at least $1 - 0.5\epsilon/n'$ by setting l to be large enough. So by the union bound, the probability that for every $i \in [n']$, $\pi^{-1}(i)$ is correctly recovered, is at least $1 - 0.5\epsilon$.

Now we bound the number of corrupted bits for every y_i. The probability that the fraction of corrupted bits in y_i is at most $\delta_1/3$ is at least $1 - 1/\mathsf{poly}(n)$ by Lemma 2.3.7. Once for every block $y_i, i = 1, \ldots, n'$, the corrupted rates are at most $\delta_1/3$, we can decode correctly. By the union bound, we know the probability that this happens is at least $1 - 1/\mathsf{poly}(n)$.

The Construction can be computed by $\mathcal{AC}^0$ circuits since all components can be computed by $\mathcal{AC}^0$ circuits.

□

Note that if we set both levels of codes in our construction to be from Lemma 2.7.5 with length $\mathsf{poly}\log n$ and l to be also $\mathsf{poly}\log n$, we can get quasi-polynomially small decoding error following the same proof. The result is stated as the follows.

Theorem 2.7.8. *For any $n \in \mathbb{N}$, any $\epsilon = 2^{-\mathsf{poly}\log n}$, there exists an explicit $(n, m = \Omega(n), \rho = \Omega(1))$ stochastic binary error correcting code with decoding error ϵ, which can be computed by $\mathcal{AC}^0$ circuits.*

We can use duplicating techniques to make the decoding error to be even smaller, however with a smaller message rate.

Theorem 2.7.9. *For any $n, r \in \mathbb{N}$, there exists an $(n, m = \Omega(n/r), \rho = \Omega(1))$ stochastic binary error correcting code with decoding error $2^{-\Omega(r/\log n)}$, which can be computed by $\mathcal{AC}^0$ circuits.*

Proof Sketch. For message $x \in \{0,1\}^m$, we simply repeat every bit for r times and then apply the coding scheme in Construction 2.7.6. When decoding, we apply the circuits from Theorem 2.7.2 to decide each x_i from r symbols. The error is

$2^{-\Omega(r/\log n)}$ since we need to correctly reconstruct $\Omega(r/\log n)$ blocks to have the approximate majority being correct.

□

2.7.3 Secure broadcasting

We give a secure protocol for the multi-party broadcasting model against external adversaries. The definition is given by Definition 2.1.11. Here we briefly describe the model again.

There are n parties where every party $i \in [n]$ has a local input $x_i \in \{0,1\}^m$. After communication, they want every party to know all $x_i, i \in [n]$. Usually we assume the communication is conducted in a broadcast channel. The adversary can observe/corrupt some messages appeared in the communication but not all of them. A protocol for this model is secure in the sense that the adversary can learn almost nothing about the local inputs.

The major difference between our model and the multi-party computation model is that, in our case, all parties are honest. The adversary can only observe/corrupt a constant fraction of messages.

The model is pretty practical in real world. For example, in military, we can think of several command-centres willing to exchange their information, while there are enemy radars or receivers which can detect their information frequently (but not always, since our army will find and attack them). Or several players want to have a common guess for the lottery but do not want anybody else occasionally passing by to know their guess.

To achieve our objective, we need to use the almost t-wise independent random

permutation. A random variable $\pi : [n] \to [n]$ is an ϵ almost t-wise independent random permutation if for every t elements $i_1, \ldots, i_t \in [n]$, $(\pi(i_1), \ldots, \pi(i_t))$ has statistical distance at most ϵ from $(\pi'(i_1), \ldots, \pi'(i_t))$ where π' is a random permutation over $[n]$. Kaplan, Naor and Reingold [KNR05] give a polynomial time construction generating ϵ almost t-wise independent permutations using $O(t \log n + \log(1/\epsilon))$ random bits.

Our protocol for secure broadcasting is as follows. We assume that all parties share a small secret key. This assumption is reasonable since in our protocol the length of the shared secret key is significantly smaller than the total input length.

Protocol 2.7.10. *For any $n, m \in \mathbb{N}$, for any $i \in [n]$, let $x_i \in \{0,1\}^m$ be the input of party i. Let the security parameter be $r \in \mathbb{N}$ with $r \leq m$.*

Let $(\mathsf{RShare}_0, \mathsf{RRec}_0)$ be an $(n_0, k_0 = \delta_0 n_0)$ robust secret sharing scheme with share alphabet $\{0,1\}^{p=O(1)}$, secret length $m_0 = m = \delta n_0$ and robust parameter $d_0 = \delta_1 n_0$, by Theorem 2.7.1 for some constant $\delta, \delta_0, \delta_1$ with $\delta_0 \geq \delta_1$.

Let $(\mathsf{RShare}_1, \mathsf{RRec}_1)$ be an $(n_1, k_1 = \delta_0 n_1)$ robust secret sharing scheme with share alphabet $\{0,1\}^{p=O(1)}$, secret length $m_1 = pn_0/r = \delta n_1$ and robust parameter $d_1 = \delta_1 n_1$, by Theorem 2.7.1.

Assume that all parties have a common secret key $s \in \{0,1\}^{O(r \log(nr))}$.

The i-th party does the following.

1. *Generate a $2^{-\Omega(r)}$-almost r-wise independent random permutation π over $[nr]$ using s.*

2. *Compute the secret shares $y_i = \mathsf{RShare}_0(x_i) \in (\{0,1\}^p)^{n_0}$. Split y_i into r blocks each of length pn_0/r such that $y_i = (y_{i,1}, \ldots, y_{i,r})$.*

3. *View the communication procedure as having $[nr]$ time slots. For $j \in [r]$, on the $\pi((i-1)r+j)$'s time slot, send message $z_{i,j} = \mathsf{RShare}_1(y_{i,j})$.*

4. *For every $i \in [n]$, $j \in [r]$, compute $y_{i,j} = \mathsf{RRec}_1(z_{i,j})$, where $z_{i,j}$ is the message received in the $\pi((i-1)r+j)$'s time slot.*

5. *For every $i \in [n]$ get $y_i = (y_{i,1}, \ldots, y_{i,r})$.*

6. *For every $i \in [n]$, $x_i = \mathsf{RRec}_0(y_i)$.*

Theorem 2.7.11. *For any $n, m, r \in \mathbb{N}$ with $r \leq m$, there exists an explicit $(n, m, \alpha = \Omega(1), n2^{-\Omega(r)}, n2^{-\Omega(r)} + nm2^{-\Omega(m/r)})$ secure broadcasting protocol with communication complexity $O(nm)$, secret key length $O(r\log(nr))$.*

Proof Sketch. Let $\alpha = 0.1\delta_1^2/p$. We consider the non-adaptive adversary. For at least $1 - 0.1\delta_1$ fraction of nr blocks, the adversary can only observe/corrupt at most δ_1/p fraction of bits in one block. Because otherwise the total observed/corrupted fraction is more than $0.1\delta_1 \times \delta_1/p$.

For every block $z_{i,j}$ with at most δ_1/p fraction of bits being corrupted, $y_{i,j}$ is hidden and can be reconstructed correctly, since $(\mathsf{RShare}_1, \mathsf{RRec}_1)$ is a robust secret sharing scheme. Let $W \subseteq [nr]$ denote the set of indices of those blocks for which the adversary can tempt more than δ_1/p fraction of bits. So $\frac{|W|}{nr} \leq 0.1\delta_1$.

Let's first assume that π is a perfect random permutation.

Let $X_{i,j}$ be the indicator such that $X_{i,j} = 1$ is the event that $\pi((i-1)n+j) \in W$. Let $X_i = \sum_{j\in[r]} X_{i,j}$. Thus

$$\Pr[X_{i,j} = 1] \leq 0.1\delta_1.$$

By Lemma 2.3.7,

$$\Pr[X_i > \delta_1 r] \leq 2^{-\Omega(r)}.$$

As $(\mathsf{RShare}_0, \mathsf{RRec}_0)$ and $(\mathsf{RShare}_1, \mathsf{RRec}_1)$ are all robust, once $X_i \leq \delta_1 r$, x_i can be reconstructed correctly. Since $\delta_0 \geq \delta_1$, x_i is also hidden.

Now consider π being a $2^{-\Omega(r)}$-almost r-wise independent random permutation. Since X_i is a deterministic function of π, by Lemma 2.2.9 the statistical distance between X_i and that in the perfect random permutation case is $2^{-\Omega(r)}$. So

$$\Pr[X_i > \delta_1 r] \leq 2^{-\Omega(r)}.$$

By the union bound, the probability that $\forall i \in [n]$, x_i is hidden and can be reconstructed correctly, is at least $1 - n2^{-\Omega(r)}$.

The reconstruction error is from the two reconstruction functions RRec_0 and RRec_1. As we applied them for at most $O(nm)$ times and want them to always compute correctly, the error is at most $n2^{-\Omega(r)} + nm2^{-\Omega(m/r)}$. Note that the support of the almost r-wise independent random permutation is a subset of the set of all permutations [KNR05], so the reconstruction can still be done following a same analysis as the proof of Theorem 2.7.4.

The total number of bits in the transmission is $nm(p/\delta)^2 = O(nm)$.

□

Chapter 3

Edit Distance: Document Exchange Protocol and Error Correcting Code

3.1 Introduction

Given two strings x, y over some finite alphabet Σ, the edit distance between them $\mathsf{ED}(x, y)$ is defined as the minimum number of edit operations (insertions, deletions and substitutions) to change x into y. Being one of the simplest metrics, edit distance has been extensively studied due to its wide applications in different areas. For example, in natural language processing, edit distance is used in automatic spelling correction to determine possible corrections of a misspelled word; and in bioinformatics it can be used to measure the similarity between DNA sequences. In this chapter, we study the general question of recovering from errors caused by edit operations. Note that without loss of generality we can only consider insertions and deletions, since a substitution can be replace by a deletion followed by an insertion, and this at most doubles the number of operations. Thus from now on we will only be interested in insertions and deletions, and we define $\mathsf{ED}(x, y)$ to be the minimum number of such operations required to change x into y.

Insertion and deletion errors happen frequently in practice. For example, they occur in the process of reading magnetic and optical media, in genetic mutation where DNA sequences may change, and in internet protocols where some packets may get lost during routing. Another typical situation where these errors can occur is in distributed file systems, e.g., when a file is stored in different machines and being edited by different people working on the same project. These files then may have different versions that need to be synchronized to remove the edit errors. In this context, we study the following two basic problems regarding insertion and deletion errors.

- *Document exchange.* In this setting, two parties Alice and Bob each holds a string x and y, and we assume that their edit distance is bounded by some parameter k. The goal is to have Alice send a sketch to Bob based on her string x and the edit distance bound k, such that Bob can recover Alice's string x based on his string y and the sketch. Naturally, we would like to require both the message length and the computation time of Alice and Bob to be as small as possible.

- *Error correcting codes.* In this setting, two parties Alice and Bob are linked by a channel where the number of worst case insertion and deletions is bounded by some parameter k. Given any message, the goal is to have Alice send an encoding of the message to Bob through the channel, so that despite any possible insertion and deletion errors that may happen, Bob can recover the correct message after receiving the (possibly modified) codeword. Again, we would like to minimize both the codeword length (or equivalently, the number of redundant bits) and the encoding/decoding time. This is a generalization of

the classical error correcting codes for Hamming errors.

It can be seen that these two problems are closely related. In particular, a solution to the document exchange problem can often be used to construct an error correcting code for insertion and deletion errors. In this chapter, we focus on the setting where the strings have a binary alphabet, arguably the most popular and important setting in computer science.[1] In this case, assume that Alice's string (or the message she wants to send) has length n, then it is known that for small k (e.g., $k \leq n/4$) both the optimal sketch size in document exchange and the optimal number of redundant bits in an error correcting code is $\Theta(k \log(\frac{n}{k}))$, and this is true even for Hamming errors. In addition, both optimum can be achieved using exponential time, with the first one using a greedy coloring algorithm and the second one using a greedy sphere packing algorithm (which is essentially what gives the Gilbert-Varshamov bound).

It turns out that in the case of Hamming errors, both optimum (up to constants) can also be achieved efficiently in polynomial time. This is done by using sophisticated linear Algebraic Geometric codes [HVLP98]. As a special case, one can use Reed-Solomon codes to achieve $O(k \log n)$ in both problems. However, the situation becomes much harder once we switch to edit errors, and our understanding of these two basic problems lags far behind the case of Hamming errors. We now survey related previous work below.

Document exchange. Historically, Orlitsky [Orl91] was the first one to study the document exchange problem. His work gave protocols for generally correlated strings

[1] Although, our document exchange protocols can be easily extended to larger alphabets, we omit the details here.

x, y using a greedy graph coloring algorithm, and in particular he obtained a deterministic protocol with sketch size $O(k \log n)$ for edit errors. However, the running time of the protocol is exponential in k. The main question left there is whether one can design a document exchange protocol that is both communication efficient and time efficient.

There has been considerable progress afterwards [Cor+00], [IMS05], [Jow12]. Specifically, Irmak et al. [IMS05] gave a randomized protocol that achieves sketch size $O(k \log(\frac{n}{k}) \log n)$ and running time $\tilde{O}(n)$. Independently, Jowhari [Jow12] also obtained a randomized protocol with sketch size $O(k \log^2 n \log^* n)$ and running time $\tilde{O}(n)$. A recent work by Chakraborty et al. [CGK16] introduced a clever randomized embedding from the edit distance metric to the Hamming distance metric, and thus obtained a protocol with sketch size $O(k^2 \log n)$ and running time $\tilde{O}(n)$. Using the embedding in [CGK16], Belazzougui and Zhang [BQ16] gave an improved randomized protocol with sketch size $O(k(\log^2 k + \log n))$ and running time $\tilde{O}(n + \text{poly}(k))$, where the sketch size is asymptotically optimal for $k = 2^{O(\sqrt{\log n})}$.

All of the above protocols, except the exponential time protocol of Orlitsky [Orl91], are however randomized. In practice, a deterministic protocol is certainly more useful than a randomized one. Thus one natural and important question is whether one can construct a deterministic protocol for document exchange with small sketch size (e.g., polynomial in $k \log n$) and efficient computation. This question is also important for applications in error correcting codes, since a randomized document exchange protocol is not very useful in designing such codes. It turns out that this question is quite tricky, and no such deterministic protocols are known even for $k > 1$ until the work of Belazzougui [Bel15] in 2015, where he gave a deterministic protocol with

sketch size $O(k^2 + k\log^2 n)$ and running time $\tilde{O}(n)$.

Error correcting codes. Error correcting codes are fundamental objects in both theory and practice. Starting from the pioneering work of Shannon, Hamming and many others, error correcting codes have been intensively studied in the literature. This is true for both standard Hamming errors such as symbol corruptions and erasures, and edit errors such as insertions and deletions. While the study of codes against standard Hamming errors has been a great success, leading to a near complete knowledge and a powerful toolbox of techniques together with explicit constructions that match various bounds, our understanding of codes for insertion and deletion errors (insdel codes for short) is still rather poor. Indeed, insertion and deletion errors are strictly more general than Hamming errors, and the study of codes against such errors has resisted progress for quite some time, as demonstrated by previous work which we discuss below.

Edit Errors Since insertion and deletion errors are strictly more general than Hamming errors, all the upper bounds on the rate of standard codes also apply to insdel codes. Moreover, by using a similar sphere packing argument, similar lower bounds on the rate (such as the Gilbert-Varshamov bound) can also be shown. In particular, one can show (e.g., [Lev66]) that for binary codes, to encode a message of length n against k insertion and deletion errors with $k \leq n/2$, the optimal number of redundant bits is $\Theta(k\log(\frac{n}{k}))$; and to protect against ε fraction of insertion and deletion errors, the optimal rate of the code is $1 - \Theta(\varepsilon\log(\frac{1}{\varepsilon}))$. On the other hand, if the alphabet of the code is large enough, then one can potentially recover from an error fraction approaching 1 or achieve the singleton bound: a rate $1-\varepsilon$ code that can correct ε

fraction of insertion and deletion errors.

However, achieving these goals have been quite challenging. In 1966, Levenshtein [Lev66] first showed that the Varshamov-Tenengolts code [RRV65] can correct one deletion with roughly $\log n$ redundant bits, which is optimal. Since then many constructions of insdel codes have been given, but all constructions are far from achieving the optimal bounds. In fact, even correcting two deletions requires $\Omega(n)$ redundant bits, and even the first explicit asymptotically good insdel code (a code that has constant rate and can also correct a constant fraction of insertion and deletion errors) over a constant alphabet did not appear until the work of Schulman and Zuckerman in 1999 [SZ99], who gave such a code over the binary alphabet. We refer the reader to the survey by Mercier et al. [MBT10] for more details about the extensive research on this topic.

In the past few years, there has been a series of work trying to improve the situation for both the binary alphabet and larger alphabets. Specifically, for larger alphabets, a line of work by Guruswami et. al [GW17], [GL16], [BG16] constructed explicit insdel codes that can correct $1-\varepsilon$ fraction of errors with rate $\Omega(\varepsilon^5)$ and alphabet size $\mathsf{poly}(1/\varepsilon)$; and for a fixed alphabet size $t \geq 2$ explicit insdel codes that can correct $1-\frac{2}{t+1}-\varepsilon$ fraction of errors with rate $(\varepsilon/t)^{\mathsf{poly}(1/\varepsilon)}$. These works aim to tolerate an error fraction approaching 1 by using a sufficiently large alphabet size. Another line of work by Haeupler et al [HS17], [HS18], [Che+18] introduced and constructed a combinatorial object called *synchronization string*, which can be used to transform standard error correcting codes into insdel codes, at the price of increasing the alphabet size. Using explicit constructions of synchronization strings, [HS17] achieved explicit insdel codes that can correct δ fraction of errors with rate $1-\delta-\varepsilon$

(hence approaching the singleton bound), although the alphabet size is exponential in $\frac{1}{\varepsilon}$.

For the binary alphabet, which is the focus of this chapter, it is well known that no code can tolerate an error fraction approaching 1 or achieve the singleton bound. Instead, the major goal here is to construct explicit insdel codes for some small fraction (ε) or some small number (k) of errors that can achieve the optimal rate of $1 - \Theta(\varepsilon \log(\frac{1}{\varepsilon}))$ or the optimal redundancy of $\Theta(k \log(\frac{n}{k}))$, which is analogous to achieving the Gilbert-Varshamov bound for standard error correcting codes. Slightly less ambitiously, one can ask to achieve redundancy $O(k \log n)$, which is optimal when $k \leq n^{1-\alpha}$ for any constant $\alpha > 0$, and easy to achieve in the case of Hamming errors by using Reed-Solomon codes. In this context, Guruswami et. al [GW17], [GL16] constructed explicit insdel codes that can correct ε fraction of errors with rate $1 - \tilde{O}(\sqrt{\varepsilon})$, which is the best possible by using code concatenation. For any fixed constant k, another work by Brakensiek et. al [BGZ17] constructed an explicit insdel code that can encode an n-bit message against k insertions/deletions with $O(k^2 \log k \log n)$ redundant bits, which is asymptotically optimal when k is a fixed constant. We remark that the construction in [BGZ17] only works for constant k, and does not give anything when k becomes larger (e.g., $k = \log n$). Finally, using his deterministic document exchange protocol, Belazzougui [Bel15] constructed an explicit insdel code that can encode an n-bit message against k insertions/deletions with $O(k^2 + k \log^2 n)$ redundant bits. In summary, there remains a huge gap between the known constructions and the optimal bounds in the case of a binary alphabet. In particular, even achieving $O(k \log n)$ redundancy has been far out of reach.

Block insertions/deletions and Transpositions We also consider document exchange protocols and error correcting codes for block edit errors, as well as block transpositions. This kind of errors is also pretty common, as most errors that happen in practice, such as in wireless or mobile communications and magnetic disk readings, tend to be concentrated. We model such errors as *block* insertions and deletions, where in one operation the adversary can insert or delete a whole block of bits. It is again easy to see that this is indeed a generalization of standard edit errors. However, in addition to the bound k on such operations, we also need to put a bound on the total number of bits that the adversary can insert or delete, since otherwise the adversary can simply delete the whole string in one block deletion. Therefore, we model the adversary as follows.

For some parameters k and t and an alphabet Σ, a (k,t) block edit adversary is allowed to perform three kinds of operations: block insertion, block deletion and block transposition. The adversary is allowed to perform at most k such operations, while the total number of symbols inserted/deleted by the first two operations is at most t. We also use (k,t) block edit errors to denote errors introduced by such an adversary. All our results focus on the case of binary alphabet, but in our protocols and analysis we will be using larger alphabets.

We note that by the result of Schulman and Zuckerman [SZ99], to correct $\Omega(n/\log n)$ block transpositions one needs at least $\Omega(n)$ redundant bits. Thus we only consider $k \leq \alpha n/\log n$ for some constant $0 < \alpha < 1$. Similarly, we only consider $t \leq \beta n$ for some constant $0 < \beta < 1$ since otherwise the adversary can simply delete the whole string. We also note the following subtle difference between the three block edit operations. While we need a bound t on the total number of

bits that the adversary can insert or delete, for block transposition an adversary can choose to move an *arbitrarily long* substring. Therefore, we need to consider the three operations separately, and cannot simply replace a block transposition by a block deletion followed by a block insertion.

Edit errors with block transpositions have been studied before in several different contexts. For example, Shapira and Storer [SS02] showed that finding the distance between two given strings under this metric is NP-hard, and they gave an efficient algorithm that achieves $O(\log n)$ approximation. Interestingly, a work by Cormode and Muthukrishnan [CM07] showed that this metric can be embedded into the L_1 metric with distortion $O(\log n \log^* n)$; and they used it to give a near linear time algorithm that achieves $O(\log n \log^* n)$ approximation for this distance, something currently unknown for the standard edit distance. Coming back to document exchange and error correcting codes, in our model, we show in the appendix that non-explicitly, the information optimum for both the sketch size of document exchange, and the redundancy of error correcting codes, is $\Theta(k \log n + t)$.

When it comes to more general errors such as block transpositions, as far as we know, there are no known explicit deterministic document exchange protocols. The only known randomized protocols which can handle edit errors as well as block transpositions are the protocol of [IMS05], which has sketch size $O(k \log(\frac{n}{k}) \log n)$; and the protocol of [Jow12], which has sketch size $\tilde{O}(k \log^2 n)$. The protocol of [IMS05] uses a recursive tree structure and random hash functions, while the protocol of [Jow12] is based on the embedding of Cormode and Muthukrishnan [CM07]. We stress that both of these protocols are randomized, and there are very good reasons why it is not easy to modify them into deterministic ones. Specifically a direct

derandomization of the hash functions used in [IMS05] (for example by using almost k-wise independent sample space) does *not* give a deterministic protocol, because block transpositions will make the computation of a matching problematic. We shall discuss this in more details when we give an overview of our techniques. On the other hand, the embedding of Cormode and Muthukrishnan [CM07] results in an exponentially large dimension, thus directly sending a sketch deterministically will result in a prohibitively large size. This is why the protocol of [Jow12] has to perform a dimension reduction first, which is necessarily randomized.

Similarly, the only previous explicit codes that can handle edit errors as well as block transpositions are the work of Schulman and Zuckerman [SZ99], and the work of Haeupler et al. [HS18]. Both can recover from $\Omega(n/\log n)$ block transpositions with $\Omega(n)$ redundant bits ([HS18] can also recover from block replications), but [SZ99] has a binary alphabet while [HS18] has a constant size alphabet. However the work of Schulman and Zuckerman [SZ99] also needs $\Omega(n)$ redundant bits even to correct one block transposition. We further note that by combining the techniques in [HS18] and [HSV17], one can get an explicit binary code that corrects k block transpositions with $\widetilde{O}(\sqrt{kn})$ redundant bits. However to our knowledge this result has not appeared anywhere in the literature, and moreover it requires at least $\widetilde{\Omega}(\sqrt{n})$ redundant bits even to correct one block transposition. We note that however none of the previous works mentioned studied edit errors that can allow block insertions/deletions.

3.1.1 Our results

In this chapter we significantly improve the situation for both document exchange and binary insdel codes. Our new constructions of explicit insdel codes are actually

almost optimal for a wide range of error parameters k. First, we have the following theorem which gives an improved deterministic document exchange protocol.

Theorem 3.1.1. *There exists a single round deterministic protocol for document exchange with communication complexity (sketch length) $O(k \log^2 \frac{n}{k})$, time complexity $\mathsf{poly}(n)$, where n is the length of the string and k is the edit distance upper bound.*

Note that this theorem significantly improves the sketch size of the deterministic protocol in [Bel15], which is $O(k^2 + k \log^2 n)$. In particular, our protocol is interesting for k up to $\Omega(n)$ while the protocol in [Bel15] is interesting only for $k < \sqrt{n}$.

Then we use this theorem to get improved binary insdel codes for the general case of k errors.

Theorem 3.1.2. *For any $n, k \in \mathsf{N}$ with $k \leq n/4$, there exists an explicit binary error correcting code with message length n, codeword length $n + O(k \log^2 \frac{n}{k})$ that can correct up to k edit errors.*

Remark 3.1.3. *In all our insdel codes, both the encoding function and the decoding function run in time $\mathsf{poly}(n)$.*

We also construct explicit document exchange protocols, and error correcting codes for block edit errors.

Theorem 3.1.4. *There exist constants $\alpha, \beta \in (0,1)$ such that for every $n, k, t \in \mathbb{N}$ with $k \leq \alpha n / \log n, t \leq \beta n$, there exists an explicit binary document exchange protocol with sketch size $O((k \log n + t) \log^2 \frac{n}{k \log n + t})$, against a (k,t) block edit adversary.*

This is the first explicit binary document exchange protocol for block edit errors. The sketch size matches the randomized protocols of [IMS05] and [Jow12] up to an additional $\log \frac{n}{k \log n + t}$ factor, and is optimal up to an additional $\log^2 \frac{n}{k \log n + t}$ factor. Using this protocol, we can construct the following error correcting code.

Theorem 3.1.5. *There exist constants $\alpha, \beta \in (0,1)$ such that for every $n, k, t \in \mathbb{N}$ with $k \leq \alpha n / \log n, t \leq \beta n$, there exists an explicit binary error correcting code with message length n and codeword length $n + O((k \log n + t) \log^2 \frac{n}{k \log n + t})$, against a (k, t) block edit adversary.*

3.1.2 Overview of the techniques

For normal edit errors. In this section we provide a high level overview of the ideas and techniques used in our constructions. We start with the document exchange protocol.

Document exchange. Our starting point is the randomized protocol by Irmak et al. [IMS05], which we refer to as the IMS protocol. The protocol is one round, but Alice's algorithm to generate the message proceeds in $L = O(\log(\frac{n}{k}))$ levels. In each level Alice computes some sketch about her string x, and her final message to Bob is the concatenation of the sketches. After receiving the message, Bob's algorithm also proceeds in h levels, where in each level he uses the corresponding sketch to recover part of x.

More specifically, in the first level Alice divides her string into $\Theta(k)$ blocks where each block has size $O(\frac{n}{k})$, and in each subsequent level every block is divided evenly into two blocks, until the final block size becomes $O(\log n)$. This takes $O(\log(\frac{n}{k}))$

levels. Using shared randomness, in each level Alice picks a set of random hash functions, one for each block which outputs $O(\log n)$ bits, and computes the hash values. In the first level, Alice's sketch is just the concatenation of the $O(k)$ hash values. In all subsequent levels, Alice obtains the sketch in this level by computing the redundancy of a systematic error correcting code (e.g., the Reed-Solomon code) that can correct $O(k)$ erasures and symbol corruptions, where each symbol has $O(\log n)$ bits (the hash value). Note that this sketch has size $O(k \log n)$ and thus the total sketch size is $O(k \log n \log(\frac{n}{k}))$.

On Bob's side, he always maintains a string $\tilde{x}$ which is the partially corrected version of x. Initially $\tilde{x}$ is the empty string, and in each level Bob tries to use his string y to fill $\tilde{x}$. This is done as follows. In each level Bob first tries to recover all the hash values of Alice in this level (notice that the hash values of the first level are directly sent to Bob). Suppose Bob has successfully recovered all the hash values, Bob then tries to match every block of Alice's string in this level with one substring of the same length in his string y, by finding such a substring with the same hash value. We say such a match is bad if the substring Bob finds is not the same as Alice's block (i.e., a hash collision). The key idea here is that if the hash functions output $O(\log n)$ bits, and they are chosen independently randomly, then with high probability a bad match only happens if the substring Bob finds contains at least one edit error. Moreover, Bob can find at least $l_i - k$ matches, where l_i is the number of blocks in the current level. Bob then uses the matched substrings to fill the corresponding blocks in $\tilde{x}$, and leaves the unmatched blocks blank. From the above discussion, one can see that there are at most k unmatched blocks and at most k mismatched blocks. Therefore in the next level when both parties divide every block evenly into two blocks, x and $\tilde{x}$ have at

most $4k$ different blocks. This implies that there are also at most $4k$ different hash values in the next level, and hence Bob can correctly recover all the hash values of Alice using the redundancy of the error correcting code.

Our deterministic protocol for document exchange is a derandomized version of the IMS protocol, with several modifications. First, we observe that the IMS protocol as we presented above, can already be derandomzied. This is because that to ensure a bad match only happens if the substring Bob finds contains at least one edit error, we in fact just need to ensure that under any hash function, no *block* of x can have a collision with a *substring* of the same length in x itself. We emphasize one subtle point here: Alice's hash function is applied to a block of her string x, while when trying to fill $\tilde{x}$, Bob actually checks every substring of the string y. Therefore we need to consider hash collisions between blocks of x and substrings of x. If the hash functions are chosen independently uniformly, then such a collision happens with probability $1/\mathsf{poly}(n)$, and thus by a union bound with high probability no collision happens. However, notice that if we write out the outputs of all hash functions on all inputs, then any collision is only concerned with two outputs which consists of $O(\log n)$ bits. Thus it's enough to use $O(\log n)$-wise independence to generate these outputs. To further save the random bits used, we can instead use an almost κ-wise independent sample space with $\kappa = O(\log n)$ and error $\varepsilon = 1/\mathsf{poly}(n)$. Using for example the construction by Alon et. al. [Alo+92b], this results in a total of $O(\log n)$ random bits (the seed), and thus Alice can exhaustively search for a fixed set of hash functions in polynomial time. Now in each level, Alice's sketch will also include the specific seed that is used to generate the hash functions, which has $O(\log n)$ bits. Note this only adds $O(\log n \log \frac{n}{k})$ to the final sketch size. Bob's algorithm is essentially the same,

except now in each level he needs to use the seed to compute the hash functions.

The above construction gives a deterministic document exchange protocol with sketch size $O(k \log n \log \frac{n}{k})$, but our goal is to further improve this to $O(k \log^2 \frac{n}{k})$. The key idea here is to use a relaxed version of hash functions with nice "self matching" properties. To motivate our construction, first observe that in each level, when Bob tries to match every block of Alice's string with one substring of the same length in his string y, it is not only true that Bob can find a matching of size at least $l_i - k$ (where l_i is the number of blocks in this level), but also true that Bob can find a *monotone* matching of at least this size. A monotone matching here means a matching that does not have edges crossing each other. In this monotone matching, there are at most k bad matches caused by edit errors, and thus there exists a *self matching* between x and itself with size at least $l_i - 2k$. In the previous construction, we in fact ensure that all these $l_i - 2k$ matches are correct. To achieve better parameters, we instead relax this condition and only require that at most k of these self matches are bad. Note if this is true then the total number of different blocks between x and $\tilde{x}$ is still $O(k)$ and we can again use an error correcting code to send the redundancy of hash values in the next level.

This relaxation motivates us to introduce *ε-self matching hash functions*, which is similar in spirit to ε-self matching strings introduced in [HS17]. Formally, we have the following definitions.

Definition 3.1.6. *(monotone matching) For every $n, n', t, p, q \in \mathbb{N}, q \leq p$, any hash functions $h_1, h_2, \ldots, h_{n'}$ where $\forall i \in [n'], h_i : \{0,1\}^p \to \{0,1\}^q$, given two strings $x \in (\{0,1\}^p)^{n'}$ and $y \in \{0,1\}^n$, a monotone matching of size t between x, y under hash functions $h_1, \ldots, h_{n'}$ is a sequence of pairs of indices*

$w = ((i_1, j_1), (i_2, j_2), \ldots, (i_t, j_t)) \in ([n'] \times [n])^t$ *s.t.* $i_1 < i_2 < \cdots < i_t$, $j_1 + p - 1 < j_2, \ldots, j_{t-1} + p - 1 < j_t, j_t + p - 1 \leq n$ *and* $\forall l \in [t], h_{i_l}(x[i_l]) = h_{i_l}(y[j_l, j_l + p - 1])$. *When* $h_1, \ldots, h_{n'}$ *are clear from the context, we simply say that* w *is a monotone matching between* x, y.

For $l \in [t]$, *if* $x[i_l] = y[j_l, j_l + p - 1]$, *we say* (i_l, j_l) *is a good match, otherwise we say it is a bad match. We say* w *is a correct matching if all matches in* w *are good. We say* w *is a completely wrong matching is all matches in* w *are bad.*

If x *and* y *are the same in terms of their binary expression, then* w *is called a self-matching.*

For simplicity, in the rest of the chapter, when we say a matching w we always mean a monotone matching.

Definition 3.1.7. *(ε-self matching hash function) Let* $p, q, n, n' \in \mathbb{N}$ *be such that* $n = n'p$. *For any* $0 < \varepsilon < 1$ *and* $x \in (\{0,1\}^p)^{n'}$, *we say that a sequence of hash functions* $h_1, h_2, \ldots, h_{n'}$ *where* $\forall i \in [n'], h_i : \{0,1\}^p \to \{0,1\}^q$ *is a sequence of* ε*-self matching hash functions with respect to* x, *if any matching between* x *and* $y \in \{0,1\}^n$ *under* $h_1, h_2, \ldots, h_n$, *where* y *is the binary expression of* x, *has at most* εn *bad matches.*

The advantage of using ε-self matching hash functions is that the output range of the hash functions can be reduced. Specifically, we can show that a sequence of ε-self matching hash functions exists with output range $\text{poly}(1/\varepsilon)$ (i.e., $O(\log(1/\varepsilon))$ bits) when the block size is at least $c \log(1/\varepsilon)$ bits for some constant $c > 1$. Furthermore, we can generate such a sequence of ε-self matching hash functions with high probability by again using an almost κ-wise independent sample space with $\kappa = O(kb_i)$,

where b_i is the current block length, and error $\varepsilon = 1/\mathsf{poly}(n)$. The idea is that in a monotone matching with εn bad matches, we can divide the matching gradually into small intervals such that at least one small interval will have the same fraction of bad matches. Thus in order to ensure the ε-self matching property we just need to make sure every small interval does not have more than ε fraction of bad matches, and this is enough by using the almost κ-wise independent sample space.

As discussed above, we need to ensure that there are at most k bad matches in a self matching, thus we set $\varepsilon = \frac{k}{n}$. Consequently now the output of the hash functions only has $O(\log(n/k))$ bits instead of $O(\log n)$ bits. Now in each level, in order to get optimal sketch size, instead of using the Reed-Solomon code we will be using an Algebraic Geometric code [HVLP98] which has redundancy $O(k \log \frac{n}{k})$. The almost κ-wise independent sample space in this case again uses only $O(\log n)$ random bits, so in each level Alice can exhaustively search the correct hash functions in polynomial time and include the $O(\log n)$ bits of description in the sketch. This gives Alice's algorithm with total sketch size $O(k \log^2 \frac{n}{k})$. On Bob's side, we need another modification: in each level after Bob recovers all the hash values, instead of simply searching for a match for every block, Bob runs a dynamic programming to find the longest monotone matching between his string y and the sequence of hash values. He then fills the blocks of $\tilde{x}$ using the corresponding substrings of matched blocks.

Error correcting codes. Our deterministic document exchange protocol can be used to directly give an insdel code for k edit errors. The idea is that to encode an n-bit message x, we can first compute a sketch of x with size r, and then encode the small sketch using an insdel code against $4k$ edit errors. Since the sketch size is

larger than k, we can use an asymptotically good code such as the one by Schulman and Zuckerman [SZ99], which results in an encoding size of $n_0 = O(r)$. The actual encoding of the message is then the original message concatenated with the encoding of the sketch.

To decode, we can first obtain the sketch by looking at the last $n_0 - k$ bits of the received string. The edit distance between these bits and the encoding of the sketch is at most $4k$, and thus we can get the correct sketch from these bits. Now we look at the bits of the received string from the beginning to index $n + k$. The edit distance between these bits and x is at most $3k$, thus if r is a sketch for $3k$ edit errors then we will be able to recover x by using r. This gives our insdel code with redundancy $O(k \log^2 \frac{n}{k})$.

For block insertions/deletions and transpositions For this kind of errors, it seems that the recently introduced synchronization techniques i.e. synchronization strings [HS17] and the technique above do not work. The reason is that synchronization strings are designed for relatively large alphabets (e.g., constant size), and often result in worse parameters when translating into the binary alphabet; while self-matching hash functions are specifically tailored for standard edit errors, and they break down once block transpositions are allowed. Instead, for document exchange we rely on the basic recursive tree structure used in [IMS05], together with a new and more sophisticated way to *approximate* maximum *non-monotone, non-overlapping* matchings in the computation; and for error correcting codes we deploy a similar strategy as for normal edit errors. We start giving more details by describing our document exchange protocol.

Document exchange. Our starting point is to generalize our protocol for normal edit errors. However, one immediate difficulty is how to handle block transpositions. The previous protocol actually performs badly for such errors. To see this, consider the following example: the adversary simply moves the first $0.4n$ bits of x to the end. Since the previous protocol tries to find the maximum monotone matching in each level, Bob can only recover the last $0.6n$ bits of x since this gives the maximum monotone matching. In this case, one single error has cost roughly half of the string; while as a comparison, for standard edit errors, Bob can recover all except $O(1)$ blocks if there is only one edit error.

To resolve this issue, we make several important changes to the previous protocol. The first major change is that, in each level, instead of having Bob find the maximum *monotone* matching between x and y using the hash values, we let Bob find the maximum *non-monotone* matching. However, the self-matching hash functions are not suitable for this purpose, since the hash functions there actually allow a small number of collisions in the hash values of blocks of x, and the use of such hash functions relies crucially on the property of a monotone matching. Instead, here we strengthen the hash function to ensure that there is no collision, by using a slightly larger output size. We call such hash functions *collision free* hash functions.

Definition 3.1.8 (Collision free hash functions)**.** *Given $n, p, q \in \mathbb{N}, p \leq n$ and a string $x \in \{0,1\}^n$, we say a function $h : \{0,1\}^p \to \{0,1\}^q$ is collision free (for x), if for every $i, j \in [n-p+1]$, $h(x[i, i+p)) = h(x[j, j+p))$ if and only if $x[i, i+p) = x[j, j+p)$. Here $x[i, j)$ denotes the substring of x which starts at the i'th bit and ends at the $j-1$'th bit.*

This definition guarantees that if the hash function we used is collision free, then

any two different substrings of x cannot have the same hash values.

We show that a collision free hash function can be constructed by using a $\frac{1}{\mathsf{poly}n}$-almost $O(\log n)$-wise independent generator with seed length (number of random bits used) $O(\log n)$. This can work since for each pair of distinct substrings, their hash values are the same with probability $1/\mathsf{poly}(n)$. Since there are at most $O(n^2)$ pairs, a union bound shows the existence of collision free hash functions. To get a deterministic hash function, we check each possible seed to see if the corresponding hash function is collision free, which can be done by checking if every pair of different substrings of x have different hash values. Note that there are at most $O(n^2)$ pairs and the seed length of the generator is $O(\log n)$, so this can be done in polynomial time.

However, even a non-monotone matching under collision free hash functions is not enough for our purpose. The reason is that in the matching, we are trying to match every well divided block of x to every possible block of y (not necessarily the blocks obtained by dividing y evenly into disjoint blocks), because we have edit errors here. If we just do this in the naive way, then the matched blocks of y can be *overlapping*. Using these overlapping blocks of y to fill the blocks of $\tilde{x}$ is problematic, since even a single edit error or block transposition can create many new (overlapping) blocks in y (which can be as large as the length of the block in each level). These new blocks are all possible to be matched, and then we won't be able to maintain an upper bound of $O(k)$ on the different blocks between x and $\tilde{x}$.

To solve this, we need to insist on computing a maximum *non-overlapping*, non-monotone matching.

Definition 3.1.9 (Non-overlapping (non-monotone) matching)**.** *Given* $n, n', p, q \in \mathbb{N}, p \leq n, p \leq n'$, *a function* $h : \{0,1\}^p \to \{0,1\}^q$ *and two strings* $x \in \{0,1\}^n, y \in$

$\{0,1\}^{n'}$, *a (non-overlapping) matching between* x *and* y *under* h *is a sequence of matches (pairs of indices)* $w = ((i_1, j_1), \ldots (i_{|w|}, j_{|w|}))$ *s.t.*

- *for every* $k \in [|w|]$,
  - $i_k = 1 + pl_k \in [n]$ *for some* l_k*, i.e., each* i_k *is the starting index of some block of* x*, when* x *is divided evenly into disjoint blocks of length* p*,*
  - $j_k \in [n']$,
  - $h(x[i_k, i_k + p)) = h(y[j_k, j_k + p))$.
- $i_1, \ldots, i_{|w|}$ *are distinct.*
- *Intervals* $[j_k, j_k + p), k \in [|w|]$*, are disjoint.*

Under this definition, we can indeed show a similar upper bound on the number of different blocks between x and $\tilde{x}$ in each level, if Bob finds the maximum non-overlapping matching. However, another technical difficulty arises: how to compute a maximum non-overlapping, non-monotone matching efficiently. This is unclear since the standard algorithm to compute a maximum matching only gives a possibly overlapping matching, while the dynamic programming approach only works for a monotone matching.

Computing the maximum non-overlapping, non-monotone matching turns out to be a hard task, and we were not able to find an efficient algorithm that accomplishes this exactly. Instead, we consider an algorithm that *approximates* the maximum non-overlapping, non-monotone matching. However, this raises several other issues. The first issue is how to maintain the invariance that in each level x and $\tilde{x}$ only differ in a small number of blocks. For example, consider level i and assuming x is

partitioned into l_i blocks, then we would like Bob to obtain a matching of size at least $l_i - O(k + t/\log n)$ (recall t is the total number of bits inserted or deleted). Thus if k and t are small then even a 0.99 approximation is still far from achieving our goal.

To get around this, we modify the protocol so that in each level Bob only computes a matching for the blocks that are unmatched in the previous level or detected to be incorrectly matched in this level (the detection can be done by comparing the hash values of the block and its matched block). If the number of such blocks can be bounded by some $O(k + t/\log n)$, then we only need a constant factor approximation. To keep the invariance in each level, note that the approximation factor should be larger than $1/2$ since each unmatched or incorrectly matched block will become two blocks in the next level.

Unfortunately, directly achieving such an approximation still seems hard. Thus we further relax the problem to allow some slight overlaps in the matching, i.e., we require that each bit of Bob's string y appears in at most d matched pairs in each level for some small number d (e.g., a constant or $\log n$). We call this a *degree d overlapping* matching (note that a non-overlapping matching is simply a degree 1 overlapping matching). Although this may cause extra errors in the matching, we show that the number of incorrectly matched pairs can be bounded by $O((k + t/\log n)i)$ (instead of $O(k + t/\log n)$) in level i.

To achieve this, we first give a $1/3$ approximation algorithm for the maximum non-monotone, non-overlapping matching. Then we give another algorithm that achieves matching size at least $2/3$ of the maximum non-monotone, non-overlapping matching, while this matching obtained is a degree 3 overlapping matching. For simplicity we also refer to this as a $2/3$ approximation algorithm.

The $1/3$ approximation is obtained by a greedy algorithm, which starts with an empty matching w and visits x's blocks one by one and tries to match it with a substring in y (according to the hash function and hash values), such that the substring does not overlap with any substring in y that is already matched. If such a matched pair is found then it is added to w. The algorithm keeps running until it cannot add any more matched pair.

To see this indeed gives a $1/3$ approximation, assume the maximum non-monotone, non-overlapping matching is w^*. Each time the algorithm adds a matched pair to w, at most 3 matched pairs in w^* will be excluded from being added to w since they either have overlaps with y's substring in the added pair or correspond to the same block of x. As a result, when $|w| < 1/3|w^*|$, there always exist some matched pairs in w^* that can be added to w. Thus, at the end of the algorithm, $|w| \geq 1/3|w^*|$.

Next we show a $2/3$ approximation algorithm that gives a degree 3 overlapping matching. The idea is to run the greedy algorithm for 3 times, where each time the algorithm is applied to unmatched blocks of x and the entire string y. To see the approximation factor, again let w^* be the optimal non-monotone, non-overlapping matching. After the first time, the matching w has size at least $1/3|w^*|$. So $|w^*|$ will have at least $|w^*| - |w|$ matched pairses for unmatched blocks in x. Therefore after the second time, the size of the matching is at least $|w| + 1/3(|w^*| - |w|) \geq 5/9|w^*|$. Similarly, after the third time, the matching will have size at least $2/3|w^*|$. As the greedy algorithm is applied three times, each bit of y can appear in at most 3 matched pairses in w.

We now bound the number of incorrectly matched and unmatched blocks in each level. First we claim that each non-monotone non-overlapping matching has at most

$O(k + t/\log n)$ incorrectly matched blocks.

This is because by our definition of collision free hash function, if a pair is incorrectly matched then the substring of y must contain some edit operation applied to x, since otherwise the pair will definitely have different hash values if they are different. Thus we only need to count how many non-overlapping new substrings in y (i.e. those not equal to any substring of x) one can get after (k,t) block edit errors. One insertion or deletion of t_1 bits will create at most $O(1) + O(t/\log n)$ new substrings since the block size is always at least $\log n$. One block transposition will create at most $O(1)$ non-overlapping substrings in y that are not equal to any substring of x. So in total there are at most $O(k + t/\log n)$ new non-overlapping substrings in y. Similarly, it is easy to generalize this claim, and show that each degree d overlapping matching has at most $O(d(k + t/\log n))$ incorrectly matched blocks.

Now to bound the number of incorrectly matched blocks in level i, notice that the matching we obtained in this level is a degree $3i$ overlapping matching, since in each level we compute a degree 3 overlapping matching using the entire string y and we combine them together. Thus there are at most $O((k + t/\log n)i)$ incorrectly matched blocks.

The number of unmatched blocks can also be upper bounded by $O((k+t/\log n)i)$ using induction. For the base case, the number of blocks in the first level of Bob is at most $l_1 = O(k + t/\log n)$ so the claim holds. Now assume in level $i-1$, the number of unmatched blocks is $c_1(i-1)(k+t/\log n)$, and the number of incorrectly matched blocks is at most $c_2(i-1)(k+t/\log n)$, for some constants c_1, c_2. In level i, once Bob recovers all the correct hash values, he can detect some of the incorrectly matched blocks. Let the total number of detected blocks and unmatched blocks be

s with $s \leq (c_1 + c_2)(i-1)(k + t/\log n)$. In our algorithm, these blocks are to be rematched in level i, and following our previous argument at least $s - c_3(k + t/\log n)$ of them can be matched in the maximum non-overlapping matching for some constant c_3. By our $2/3$ approximation algorithm, the actual matching w_i we get has size at least $2/3(s - c_3(k + t/\log n))$. Hence the number of unmatched blocks after this is at most $s - |w_i| \leq 1/3s + 2/3c_3(k + t/\log n)$. We can set c_1 to be large enough s.t. this number is still upper bounded by $c_1 i(k + t/\log n)$.

As we have bounded the number of incorrectly matched blocks and unmatched blocks by $O(i(k + t/\log n))$ in level i, at the beginning of level $i+1$, Alice can send the redundancy of the hash values of her blocks using a code that corrects $O(i(k + t/\log n))$ errors. This allows Bob to recover all the hash values correctly, and the size of the redundancy is $O(i(k + t/\log n)\log n)$ since the hash function outputs $O(\log n)$ bits. We start the protocol with a block size of $O(\frac{n}{k\log n + t})$ and thus the protocol takes $L = O(\log \frac{n}{k\log n+t})$ levels. A straightforward computation gives that the sketch size of our protocol is $O((k\log n + t)\log^2 \frac{n}{k\log n+t})$.

Error correcting codes. We now describe how to construct an error correcting code from a document exchange protocol for block edit errors. Our starting point is to first encode the sketch of the document exchange protocol using the code by Schulman and Zuckerman [SZ99], which can resist edit errors and block transpositions. Then we concatenate the message with the encoding of the sketch. When decoding, we first decode the sketch, then apply the document exchange protocol on Bob's side to recover the message.

However, here we have an additional issue with this approach: a block transposition may move some part of the encoding of the sketch to somewhere in the middle of the message, or vice versa. In this case, we won't be able to tell which part of the received string is the encoding of the sketch, and which part of the received string is the original message.

To solve this issue, we use a fixed string $\mathsf{buf} = 0^{\ell_{\mathsf{buf}}} \circ 1$ as a buffer to mark the encoding of the sketch, for some $\ell_{\mathsf{buf}} = O(\log n)$. More specifically, we evenly divide the encoding of the sketch into small blocks of length ℓ_{buf}, and insert buf before every block. Note that this only increases the length of the encoding of the sketch by a constant factor. The reason we use such a small block length is that, even if the adversary can forge or destroy some buffers, the total number of bits inserted or deleted caused by this is still small. In fact, we can bound this by $O(k)$ block insertions/deletions with at most $O(k \log n)$ bits inserted/deleted, for which both the sketch and the encoding of the sketch can handle. When decoding, we first recognize all the buf's. Then we take the ℓ_{buf} bits after each buf to form the decoding of the sketch, and take the remaining bits as the message.

Unfortunately, this approach introduces two additional problems here. The first problem is that the original message may contain buf as a substring. If this happens then in the decoding procedure again we will be taking part of the message to be in the encoding of the sketch. The second problem is that the small blocks of the encoding of the sketch may also contain buf. In this case we will be deleting information from the encoding of the sketch, which causes too many edit errors.

To address the first problem, we turn the original message into a *pseudorandom* string by computing the XOR of the message with the output of an appropriate

pseudorandom generator that has seed length $O(\log n)$. We show that with high probability buf does not appear as a substring in the XOR. We can then exhaustively search for a seed that satisfies this requirement, and append the seed to the sketch of the document exchange protocol.

To address the second problem, we choose the length of the buffer to be longer than the length of each block in the encoding of the sketch, so that buf doesn't appear as a substring in any block. This is exactly why we choose the length of the buffer to be $\ell_{\mathsf{buf}} + 1$ while we choose the length of each block to be ℓ_{buf}.

We can directly apply our document exchange protocol to the construction above, obtaining an error correcting code with $O((k \log n + t) \log^2 \frac{n}{k \log n + t})$ redundant bits.

Chapter Organization In Section 3.2 we introduce some notation and basic techniques used in this chapter. In Section 3.3 we give the deterministic protocol for document exchange. In Section 3.4 we construct error correcting codes for edit errors using results from previous sections.

3.2 Preliminaries

3.2.1 Notations

Let Σ be an alphabet (which can also be a set of strings). For a string $x \in \Sigma^*$,

1. $|x|$ denotes the length of the string.
2. $x[i, j]$ denotes the substring of x from position i to position j (Both ends included).
3. $x[i]$ denotes the i-th symbol of x.

4. $x \circ x'$ denotes the concatenation of x and some other string $x' \in \Sigma^*$.

5. B-prefix denotes the first B symbols of x. (Usually used when $\Sigma = \{0,1\}$.)

6. x^N the concatenation of N number of string x.

We use U_n to denote the uniform distribution on $\{0,1\}^n$.

3.2.2 Edit distance and longest common subsequence

Definition 3.2.1 (Edit distance)**.** *For any two strings $x, x' \in \Sigma^n$, the edit distance $ED(x,x')$ is the minimum number of edit operations (insertions and deletions) required to transform x into x'.*

Definition 3.2.2 (Longest Common Subsequence)**.** *For any strings x, x' over Σ, the longest common subsequence of x and x' is the longest pair of subsequences of x and x' that are equal as strings. $LCS(x,x')$ denotes the length of the longest common subsequence between x and x'.*

Note that $ED(x,x') = |x| + |x'| - 2LCS(x,x')$.

Definition 3.2.3 (Block edit errors)**.** *A block-insertion/deletion (or burst-insertion/deletion) of b symbols to a string x is defined to be inserting/deleting a block of consecutive b symbols to x. When we do not need to specify the number of symbols inserted or deleted, we simply say a block-insertion/deletion.*

We define (k,t)-block-insertions/deletions (to x) to be a sequence of k block-insertions/deletions, where the total number of symbols inserted/deleted is at most t. Similarly, we define (k,t)-block edit errors to be a sequence of k block-insertions, deletions, and transpositions, where the total number of symbols inserted/deleted is at most t.

3.2.3 Almost k-wise independence

Definition 3.2.4 (ε-almost κ-wise independence in max norm [Alo+92b])**.** *Random variables $X_1, X_2, \ldots, X_n \in \{0,1\}^n$ are ε-almost κ-wise independent in max norm if $\forall i_1, i_2, \ldots, i_\kappa \in [n]$, $\forall x \in \{0,1\}^\kappa$, $|\Pr[X_{i_1} \circ X_{i_2} \circ \cdots \circ X_{i_\kappa} = x] - 2^{-\kappa}| \leq \varepsilon$.*

A function $g : \{0,1\}^d \to \{0,1\}^n$ is an ε-almost κ-wise independence generator in max norm if $g(U) = X = X_1 \circ \cdots X_n$ are ε-almost κ-wise independent in max norm.

In the following passage, unless specified, when we say ε-almost κ-wise independence, we mean in max norm.

Theorem 3.2.5 (ε-almost κ-wise independence generator [Alo+92b])**.** *There exists an explicit construction s.t. for every $n, \kappa \in \mathbb{N}$, $\varepsilon > 0$, it computes an ε-almost κ-wise independence generator $g : \{0,1\}^d \to \{0,1\}^n$, where $d = O(\log \frac{\kappa \log n}{\varepsilon})$.*

The construction is highly explicit in the sense that, $\forall i \in [n]$, the i-th output bit can be computed in time $\mathsf{poly}(\kappa, \log n, \frac{1}{\varepsilon})$ given the seed and i.

3.2.4 Pseudorandom generator

Definition 3.2.6 (Pseudorandom generator)**.** *A generator $g : \{0,1\}^r \to \{0,1\}^n$ is a pseudorandom generator (PRG) against a function $f : \{0,1\}^n \to \{0,1\}$ with error ε if*

$$|\Pr[f(U_n) = 1] - \Pr[f(g(U_r)) = 1]| \leq \varepsilon$$

where r is called the seed length of g.

We also say g ε-fools function f. Similarly, g ε-fools a class of function $\mathcal{F}$ if g fools all functions in $\mathcal{F}$.

Theorem 3.2.7 (PRG for CNF/DNFs [De+10]). *There exists an explicit PRG g s.t. for every $n, m \in \mathbb{N}, \varepsilon > 0$, every CNF/DNF f with n variables, m terms,*

$$|\Pr[f(U_n) = 1] - \Pr[f(g(n, m, \varepsilon, U_r)) = 1]| \leq \varepsilon$$

where $|g(n, m, \varepsilon, U_r)| = n$, $r = O(\log n + \log^2(m/\varepsilon) \cdot \log\log(m/\varepsilon))$.

3.2.5 Random walk on expander graphs

Definition 3.2.8 ((n, d, λ)-expander graphs). *If G is an n-vertex d-regular graph and $\lambda(G) \leq \lambda$ for some number $\lambda < 1$, then we say that G is an (n, d, λ)-expander graph. Here $\lambda(G)$ is defined as the second largest eigenvalue (in the absolute value) of the normalized adjacency matrix A_G of G.*

A family of graphs $\{G_n\}_{n\in\mathbb{N}}$ is a (d, λ)-expander graph family if there are some constants $d \in \mathbb{N}$ and $\lambda < 1$ such that for every n, G_n is an (n, d, λ)-expander graph.

Theorem 3.2.9 (Random walk on expander graphs, [Alo+95], [HLW06] Theorem 3.11). *Let $A_0, \ldots, A_t$ be vertex sets of densities $\alpha_0, ..., \alpha_t$ in an (n, d, λ)-expander graph G. Let $X_0, \ldots, X_t$ be a random walk on G. Then*

$$\Pr\{\forall i, X_i \in A_i\} \leq \prod_{i=0}^{t-1}(\sqrt{\alpha_i \alpha_{i+1}} + \lambda).$$

It is well known that some expander families have strongly explicit constructions.

Theorem 3.2.10 (Explicit expander family [AB09]). *For every constant $\lambda \in (0, 1)$ and some constant $d \in \mathbb{N}$ depending on λ, there exists a strongly explicit (d, λ)-expander family.*

3.2.6 Error correcting codes (ECC)

An (n, m, d)-code C is an ECC (for hamming errors) with codeword length n, message length m. The hamming distance between every pair of codewords in C is at least d.

Next we recall the definition of ECC for edit errors.

Definition 3.2.11. *An ECC $C \subseteq \{0,1\}^n$ for edit errors with message length m and codeword length n consists of an encoding mapping $Enc : \{0,1\}^m \to \{0,1\}^n$ and a decoding mapping $Dec : \{0,1\}^* \to \{0,1\}^m \cup \{Fail\}$. The code can correct k edit errors if for every y, s. t. $ED(y, Enc(x)) \leq k$, we have $Dec(y) = x$. The rate of the code is defined as $\frac{m}{n}$.*

An ECC family is explicit (or has an explicit construction) if both encoding and decoding can be done in polynomial time.

We will utilize linear algebraic geometry codes to compute the redundancy of the hamming error case.

Theorem 3.2.12 ([HVLP98])**.** *There exists an explicit algebraic geometry ECC family $\{(n, m, d)_q$-code $C \mid n, m \in \mathbb{N}, m \leq n, d = n - m - O(1), q = \mathsf{poly}(\frac{n}{d})\}$ with polynomial-time decoding when the number of errors is less than half of the distance.*

Moreover, $\forall n, m \in \mathbb{N}$, for every message $x \in \mathbb{F}_q^m$, the codeword is $x \circ z$ for some redundancy $z \in \mathbb{F}_q^{n-m}$.

To construct ECC from document exchange protocol, we need to use a previous result about asymptotically good binary ECC for edit errors given by Schulman and Zuckerman [SZ99].

Theorem 3.2.13 ([SZ99]). *There exists an explicit binary ECC family in which a code with codeword length n, message length $m = \Omega(n)$, can correct up to $k = \Omega(n)$ edit errors.*

3.3 Deterministic protocol for document exchange

We derandomize the IMS protocol given by Irmak et. al. [IMS05], by first constructing ε-self-matching hash functions and then use them to give a deterministic protocol.

3.3.1 ε-self-matching hash functions

The following describes a matching property between strings under some given hash functions.

Definition 3.3.1. *For every $n, n', t, p, q \in \mathbb{N}, q \leq p$, any hash functions $h_1, h_2, \ldots, h_{n'}$ where for every $i \in [n'], h_i : \{0,1\}^p \to \{0,1\}^q$, given two strings $x' \in (\{0,1\}^p)^{n'}$ and $y \in \{0,1\}^n$, a (monotone) matching of size t between x', y under hash functions $h_1, \ldots, h_{n'}$ is a sequence of pairs of indices $w = ((i_1, j_1), (i_2, j_2), \ldots, (i_t, j_t)) \in ([n'] \times [n])^t$ s.t. $1 \leq i_1 < i_2 < \cdots < i_t \leq n'$, $j_1 + p - 1 < j_2, \ldots, j_{t-1} + p - 1 < j_t, j_t + p - 1 \leq n$ and $\forall l \in [t], h_{i_l}(x'[i_l]) = h_{i_l}(y[j_l, j_l + p - 1])$.*

For $l \in [t]$, if $x'[i_l] = y[j_l, j_l + p - 1]$, we say (i_l, j_l) is a good pair, otherwise we say it is a bad pair. We say w is a correct matching if all pairs in w are good. We say w is a completely wrong matching is all pairs in w are bad.

If parsing x' to be binary we get x which is equal to y, then w is called a self-matching of x' (or x).

For the match w between x', y under hash functions $h_1, \ldots, h_n$, we simply say a

match w if x', y and $h_1, \ldots, h_{n'}$ are clear in the context.

The next lemma shows that the maximum matching between two strings under some hash functions can be computed in polynomial time, using dynamic programming.

Lemma 3.3.2. *There is an algorithm s.t. for every $n, n', p, q \in \mathbb{N}, q \leq p$, any hash functions $h_1, h_2, \ldots, h_{n'}$ where for every $i \in [n']$, $h_i : \{0,1\}^p \to \{0,1\}^q$, every $x' \in (\{0,1\}^p)^{n'}$ and $y \in \{0,1\}^n$, given $h_1(x'[1])$, ..., $h_{n'}(x'[n'])$, y, n, n', p, q, it can compute the maximum matching between x', y under hash functions $h_1, \ldots, h_{n'}$ in time $O(n^2(t_h + \log n))$, if for every $i \in [n']$, h_i can be computed in time t_h. (Note that x is not necessary to be part of the input).*

Proof. We present a dynamic programming to compute the maximum matching.

For every $j' \in [n'], j \in [n]$, let $f(j', j)$ be the size of the maximum matching between $x'[1, j']$ and $y[1, j]$ under $h_1, \ldots, h_{j'}$. We compute f as follows,

$$f(j', j) = \begin{cases} \max(f(j'-1, j-p)+1, f(j'-1, j), f(j', j-1)), \\ \qquad\qquad \text{if } h_{j'}(x'[j']) = h_{j'}(y[j-p+1, j]); \\ \max(f(j'-1, j), f(j', j-1)), \\ \qquad\qquad \text{if } h_{j'}(x'[j']) \neq h_{j'}(y[j-p+1, j]). \end{cases}$$

Although f only computes the size of the maximum matching, we can record the corresponding matching every time when we compute $f(j', j), j' \in [n'], j \in [n]$. So finally we can get the maximum matching after computing $f(n', n)$.

We need to compute $f(j', j), j' \in [n'], j \in [n]$ one by one and $nn' = O(n^2)$. Every time we compute an $f(j', j)$, we need to compute a constant number of evaluations of the hash functions and append a pair of indices to some previous records to create the current record of the maximum matching. That takes $O(\log n + t_h)$. So

the overall time complexity is $O(n^2(\log n + t_h))$. □

Next we show that a matching between two strings with edit distance k, induces a self-matching in one of the strings, where the number of bad pairs decreases by at most k.

Lemma 3.3.3. *For every $n, n', k, m, p, q \in \mathbb{N}, k \leq n', q \leq p$, any hash functions $h_1, h_2, \ldots, h_{n'}$ where $\forall i \in [n'], h_i : \{0,1\}^p \to \{0,1\}^q$, given two sequences $x \in \{0,1\}^{pn'}$ and $y \in \{0,1\}^n$ s.t. $\mathsf{ED}(x,y) \leq k$, parsing x to be $x' \in (\{0,1\}^p)^{n'}$, if there exists a matching w between x' and y under $h_1, \ldots, h_{n'}$ having at least m bad pairs, then there is a self-matching w' of x with size $|w'| \geq |w| - k$, having at least $m - k$ bad pairs.*

Proof. Assume w.l.o.g. the m bad pairs in the matching are the pairs $((j'_1, j_1), \ldots, (j'_m, j_m))$.

As the number of edit errors is upper bounded by k, to edit y back to x, we only need to do insertions and deletions on at most k of substrings $y[j_1, j_1 + p - 1], y[j_2, j_2 + p - 1], \ldots, y[j_m, j_m + p - 1]$. The substrings left unmodified induce a self-matching of x, which is a subsequence of w. Note that only at most k entries of w are excluded. So the number of bad pairs in the matching is at least $m - k$ and the size of w' is at least $|w| - k$.

□

The following property shows that a matching between two long intervals induces two shorter intervals having a matching s.t. the ratio between the matching size and the total interval length is maintained.

Lemma 3.3.4. *For every $n, n', t, p, q \in \mathbb{N}, t \leq n, q \leq p$, any hash functions $h_1, h_2, \ldots, h_{n'}$ where $\forall i \in [n], h_i : \{0,1\}^p \to \{0,1\}^q$, given two sequences $x' \in$*

$(\{0,1\}^p)^{n'}$ *and* $y \in \{0,1\}^n$*, if there is a matching* w *between* x *and* y *under* $h_1, \ldots, h_{n'}$ *having size* t*, then for every* $t^* \leq t$*, there is a matching* w^* *between* x'_0, y_0 *having size* t^**, where* x'_0 *is an interval of* x'*,* y_0 *is an interval of* y*, and* $|x_0| + |y_0|/p \leq \frac{2t^*}{t}(n' + n/p)$*. Here* w^* *is a subsequence of* w*.*

Proof. We consider the following recursive procedure.

At the beginning (the first round), let $w_1 = w$. We know that $n' + n/p \leq \frac{t}{t}(n' + n/p)$.

For the i-th round, assume we have a matching w_i between $x'[j'_1, j'_2]$, $y[j_1, j_2]$ with $|w_i| = t_i$ and $|x'[j'_1, j'_2] + y[j_1, j_2]| \leq \frac{t_i}{t}(n' + n/p)$. Let $w_i = (\rho'_1, \rho_2), \ldots, (\rho'_{t_i}, \rho_{t_i})$.

We pick $l = \lfloor t_i/2 \rfloor$. Consider the following two sequences $w_i[1, l]$ and $w_i[l + 1, t_i]$. Here $w_i[1, l]$ is a matching between $x'_1 = x'[j'_1, \rho'_l]$ and $y_1 = y[j_1, \rho_l + p - 1]$. Also $w_i[l + 1, t_i]$ is a matching between $x'_2 = x'[\rho'_{l+1}, j'_2]$ and $y_2 = y[\rho_l + p, j_2]$.

Note that either $(|x'_1| + |y_1|/p) \leq \frac{l}{t}(n' + n/p)$ or $(|x'_2| + |y_2|/p) \leq \frac{t_i - l}{t}(n' + n/p)$. Because if not, then

$$|x'[j'_1, j'_2]| + |y[j_1, j_2]| = (|x'_1| + |y_1|/p) + (|x'_2| + |y_2|/p) > \frac{t_i}{t}(n' + n/p),$$

which contradicts the assumption that $|x'[j'_1, j'_2] + y[j_1, j_2]| \leq \frac{t_i}{t}(n' + n/p)$. Thus we can pick one sequence as w_{i+1}.

We can go on doing this until the i^*-th round in which there is a matching w_{i^*} of size $t_{i^*} \in [t^*, 2t^*)$, whose corresponding pair of substrings are x'_0 and y_0. We know that $|x'_0| + |y_0|/p \leq \frac{t_{i^*}}{t}(n + n'/p)$. We pick t^* pairs in w_{i^*} and this gives a matching between x'_0 and y_0 s.t. $|x'_0| + |y_0|/p \leq \frac{2t^*}{t}(n' + n/p)$.

□

We now describe an explicit construction for a family of sequences of ε-self-matching hash functions.

Theorem 3.3.5. *There exists an algorithm which, on input $n, p, q \in \mathbb{N}$, $\varepsilon \in (0,1)$, $n \geq p \geq q$, $q = \Theta(\log \frac{1}{\varepsilon})$, $x \in \{0,1\}^n$, outputs a description of ε-self-matching functions $h_1, \ldots, h_{n'} : \{0,1\}^p \to \{0,1\}^q$, in time $\mathsf{poly}(n)$, where the description length is $O(\log n)$ and $n' = \frac{n}{p}$.*

Also there is an algorithm which, given the same $n, p, q \in \mathbb{N}$, the description of $h_1, \ldots, h_{n'}$, $i \in [n']$ and any $a \in \{0,1\}^p$, can output $h_i(a)$ in time $\mathsf{poly}(n)$.

To prove this theorem we consider the following construction.

Construction 3.3.6. *Let $n, p, q \in \mathbb{N}$, $\varepsilon \in (0,1)$, $n \geq p \geq q$, $q = \Theta(\log \frac{1}{\varepsilon})$, $x \in \{0,1\}^n$ be given parameters.*

1. *Divide x into consecutive blocks to get $x' \in (\{0,1\}^p)^{n'}$, $n' = n/p$;*
2. *Let $g : \{0,1\}^d \to \{0,1\}^{qn'2^p}$ be the ε_g-almost κ-wise independence generator from Theorem 3.2.5, where $\kappa = O(\varepsilon nq)$, $\varepsilon_g = 1/\mathsf{poly}(n), d = O(\log n)$;*
3. *View the output of g as in a two dimension array $(\{0,1\}^q)^{[n'] \times \{0,1\}^p}$;*
4. *Exhaustively search $u \in \{0,1\}^d$ s.t.*

   ($\star$) For every consecutive $t_1 \leq \frac{4m}{p\varepsilon}$-blocks $x'_0 \in (\{0,1\}^p)^{t_1}$ of $(x'[1], \ldots, x'[n'])$ and every substring x_0 of x having length $t_2 \leq \frac{4m}{\varepsilon}$, x'_0 and x_0 have no completely wrong matching of size $m = \Theta(\frac{\log n}{q})$ under functions $h_i(\cdot) = g(u)[i][\cdot], i \in [n']$, where $g(u)[i][\cdot]$ is the corresponding entry in the two dimension array $g(u)$;

5. *return u which is the description of the ε-self-matching hash functions.*

(Evaluation of a hash function) Given u, $i \in [n']$, n, p, q, ε, an input $v \in \{0,1\}^p$, one can compute $h_i(v) = g(u)[i][v]$.

Lemma 3.3.7. *There exists an u such that the assertion $(\star)$ holds.*

Proof. We consider a uniform random string u and show that with high probability u satisfies assertion $(\star)$.

Fix a pair of substrings $x'[j_1', j_2']$, $x[j_1, j_2]$, and a matching $w^* = ((\rho_1', \rho_1), \ldots, (\rho_m', \rho_m))$ between them of size $m = \Theta(\frac{\log n}{q})$. The probability

$$
\begin{aligned}
&\Pr_u\{w^* \text{ is a completely wrong matching }\} \\
\leq &\Pr_u\{\forall l \in [m], h_{\rho_l'}(x'[\rho_l']) = h_{\rho_\ell'}(x[\rho_\ell, \rho_\ell + p - 1])\} \\
\leq &\sum_{a \in (\{0,1\}^p)^m} \Pr_u\{\forall \ell \in [m], h_{\rho_\ell'}(x'[\rho_\ell']) = h_{\rho_\ell'}(x[\rho_\ell, \rho_\ell + p - 1]) = a_\ell\} \\
\leq &2^{qm}((\frac{1}{2^q})^{2m} + \varepsilon_g) \\
\leq &\frac{1}{2^{qm}} + 2^{qm}\varepsilon_g \\
= &\frac{1}{\mathsf{poly}(n)},
\end{aligned}
$$

as long as we take ε_g to be a sufficiently small $1/\mathsf{poly}(n)$. The total number of pairs $x'[j_1', j_2']$, $x[j_1, j_2]$ is at most $\mathsf{poly}(n)$. For each fixed pair there are at most $\binom{t_1}{m}\binom{t_2}{m} \leq (\Theta(\frac{1}{\varepsilon}))^{\Theta(\frac{\log n}{q})} = (\Theta(\frac{1}{\varepsilon}))^{\Theta(\frac{\log n}{\log \frac{1}{\varepsilon}})} = \mathsf{poly}(n)$ number of different matchings of size m. Thus by the union bound, the probability that the assertion $(\star)$ holds is at least $1 - 1/\mathsf{poly}(n)$ if we choose the parameters appropriately.

□

Lemma 3.3.8. *Construction 3.3.6 gives a sequence of ε-self-matching hash functions.*

Proof. Let w be a self-matching of x, having at least $\varepsilon n + 1$ bad pairs. We pick all the wrong pairs in this matching to get a completely wrong self-matching $\tilde{w}$.

By Lemma 3.3.4, there are two substrings, $x'[j'_1, j'_2]$ and $x[j_1, j_2]$, $1 \le j'_1 \le j'_2 \le [n'], 1 \le j_1 \le j_2 \le [n], |x'[j'_1, j'_2]| + |x[j_1, j_2]|/p \le \frac{2m}{\varepsilon n+1}(n' + n/p)$, having a completely wrong matching of size at least m, which is a subsequence of $\tilde{w}$. Note that $|x'[j'_1, j'_2]| \le \frac{2m}{\varepsilon n+1}(n' + n/p) = \frac{4mn}{(\varepsilon n+1)p} \le \frac{4m}{\varepsilon p}$. Also $|x[j_1, j_2]| \le \frac{2m}{\varepsilon n+1}(n/p + n/p)p = \frac{4mn}{\varepsilon n+1} \le \frac{4m}{\varepsilon}$. This contradicts (⋆).

□

Lemma 3.3.9. *The evaluation of a function in the sequence takes polynomial time.*

Proof. For evaluation, by Lemma 3.2.5, given a seed and a position index, the corresponding bit in g's output can be computed in time $\mathsf{poly}(\kappa, \log(qn'2^p), \frac{1}{\varepsilon_g}) = \mathsf{poly}(n)$. The output of a function has q bits. One can compute the q bits one by one and the total running time is still $\mathsf{poly}(n)$.

□

Lemma 3.3.10. *The algorithm runs in polynomial time.*

Proof. Checking the assertion (⋆) takes time $\mathsf{poly}(n)$. To see this, first note that there are $\mathsf{poly}(n)$ pairs of x'_0 and x_0. Also for each pair, the total number of matchings of size m is at most

$$\binom{t_1}{m}\binom{t_2}{m} \le \binom{\frac{4m}{p\varepsilon}}{m}\binom{\frac{4m}{\varepsilon}}{m} \le (\frac{4e}{\varepsilon})^{2m} = (\frac{4e}{\varepsilon})^{\Theta(\frac{\log n}{q})} = (\frac{4e}{\varepsilon})^{\Theta(\frac{\log n}{\log \frac{1}{\varepsilon}})} = \mathsf{poly}(n).$$

For a specific matching, Alice can first compute the hash values in time $\mathsf{poly}(n)$ by Lemma 3.3.9. Alice can check whether it is a wrong matching and compute the size of the matching in time $\mathsf{poly}(n)$.

Note that there are $\mathsf{poly}(n)$ number of different seeds. So the algorithm of generating the description runs in polynomial time.

□

Proof of Theorem 3.3.5. We use Construction 3.3.6, the theorem directly follows from Lemma 3.3.7, 3.3.8, 3.3.10.

□

3.3.2 Deterministic protocol for document exchange

Our deterministic protocol for document exchange is as follows.

Construction 3.3.11. *The protocol is for every input length $n \in \mathbb{N}$, every $k \leq \alpha n$ number of edit errors where α is a constant. For the case $k > \alpha n$, we simply let Alice send her input string.*

Both Alice's and Bob's algorithms have $L = O(\log \frac{n}{k})$ levels.

Alice: On input $x \in \{0,1\}^n$;

1. *We set up the following parameters;*
   - *For every $i \in [L]$, in the i-th level,*
     - *The block size is $b_i = \frac{n}{3 \cdot 2^i k}$, i.e., in each level we divide a block in the previous level evenly into two blocks. We choose L properly s.t. $b_L = O(\log \frac{n}{k})$;*

– *The number of blocks* $l_i = n/b_i$;

2. *For the i-th level,*

   2.1. *Divide x into consecutive blocks to get* $x' \in (\{0,1\}^{b_i})^{l_i}$;

   2.2. *Construct a sequence of* $\varepsilon = \frac{k}{n}$*-self-matching hash functions* $h_1, \ldots, h_{l_i} : \{0,1\}^{b_i} \to \{0,1\}^{b^*}$ *for x by Theorem 3.3.5, with* $b^* = O(\log \frac{n}{k})$. *Let the description of the hash functions be* $u[i] \in \{0,1\}^{O(\log n)}$ *by Theorem 3.3.5;*

   2.3. *Compute* $v[i] = (h_1(x'[1]), h_2(x'[2]), \ldots, h_{l_i}(x'[l_i]))$;

   2.4. *Compute the redundancy* $z[i] \in (\{0,1\}^{b^*})^{\Theta(k)}$ *for* $v[i]$ *by Theorem 3.2.12, where the code has distance* $14k$;

3. *Compute the redundancy* $z_{\text{final}} \in (\{0,1\}^{b_L})^{\Theta(k)}$ *for the blocks of the L-th level by Theorem 3.2.12, where the code has distance* $8k$;

4. *Send* $u = (u[1], u[2], \ldots, u[L])$, $z = (z[1], z[2], \ldots, z[L])$, $v[1]$, z_{final}.

Bob: On input $y \in \{0,1\}^{O(n)}$ *and received* u, z, $v[1]$, z_{final};

1. *Create* $\tilde{x} \in \{0,1,*\}^n$ *(i.e. his current version of Alice's x), initiating it to be* $\{*, *, \ldots, *\}$;

2. *For the i-th level where* $1 \le i \le L-1$,

   2.1. *Divide* $\tilde{x}$ *into consecutive blocks to get* $\tilde{x}' \in (\{0,1\}^{b_i})^{l_i}$;

   2.2. *Apply the decoding of Theorem 3.2.12 on* $h_1(\tilde{x}'[1]) \circ h_2(\tilde{x}'[2]) \circ \ldots \circ h_{l_i}(\tilde{x}'[l_i]) \circ z_i$ *to get the sequence of hash values* $v[i] = (h_1(x[1]), h_2(x[2]),$

$\ldots, h_{l_i}(x[l_i]))$. *Note that* $v[1]$ *is received directly, thus Bob does not need to compute it;*

2.3. *Compute* $w = ((\rho'_1, \rho_1), \ldots, (\rho'_{|w|}, \rho_{|w|})) \in ([l_i] \times [|y|])^{|w|}$ *which is the maximum matching between* x' *and* y *under* $h_1, \ldots, h_{l_i}$*, using* $v[i]$*, by Lemma 3.3.2;*

2.4. *Evaluate* $\tilde{x}$ *according to the matching, i.e. let* $\tilde{x}'[\rho'_j] = y[\rho_j, \rho_j + b_i - 1]$*;*

3. *In the* L*'th level, apply the decoding of Theorem 3.2.12 on the blocks of* $\tilde{x}$ *and* z_{final} *to get* x*;*

4. *Return* x*.*

Lemma 3.3.12. *Both Alice's and Bob's algorithms are in polynomial time.*

Proof. We first consider Alice's algorithm. For the i-th level, $i \in [L]$, dividing x into blocks takes time $O(n)$. Computing the description and doing evaluation of ε-self-matching hash functions takes time $\mathsf{poly}(n)$ by Theorem 3.3.5.

The redundancy $z[i], i \in [L]$ and z_{final} can be computed in polynomial time by Theorem 3.2.12.

Thus the time complexity for Alice is $\mathsf{poly}(n)$.

Next we consider Bob's algorithm. Creating $\tilde{x}'$ at the beginning of each level takes $O(n)$ time. Decoding of $h_1(\tilde{x}'[1]) \circ h_2(\tilde{x}'[2]) \circ \ldots \circ h_{l_i}(\tilde{x}'[l_i]) \circ z[i]$ takes $\mathsf{poly}(n)$ time by Theorem 3.2.12. By Lemma 3.3.2, computing the maximum matching between x' and y under $h_1, \ldots, h_{l_i}$ takes time $O(n^2(t_h + \log n))$ where t_h is the time complexity of evaluating any one of the hash functions. By Theorem 3.2.5, t_h is $\mathsf{poly}(n)$, so computing the maximum matching takes $\mathsf{poly}(n)$. Decoding on the last level of $\tilde{x}$'s blocks and z_{final} also takes polynomial time by Theorem 3.2.12.

So the overall running time for Bob is also $\mathsf{poly}(n)$. □

Lemma 3.3.13. *The communication complexity is* $O(k \log^2 \frac{n}{k})$.

Proof. For every $i \in [L]$, $|u[i]| = O(\log n)$ by Theorem 3.3.5. Thus the total number of bits of u is $L \cdot O(\log n) = O(\log \frac{n}{k} \log n)$.

Also for every $i \in [L]$, by Theorem 3.2.12 the number of bits in z_i is $O(k \log \frac{n}{k})$. So the total number of bits of z is $O(k \log^2 \frac{n}{k})$.

Again by Theorem 3.2.12, z_{final} has $O(k \log \frac{n}{k})$ bits.

Note that $v[1]$ has $O(k \log \frac{n}{k})$ bits, since there are $l_1 = O(k)$ blocks in the first level and the length of each hash value is $O(\log \frac{n}{k})$.

Thus the total communication complexity is $O(\log \frac{n}{k} \log n + k \log^2 \frac{n}{k}) = O(k \log^2 \frac{n}{k})$, since $k \log \frac{n}{k} \geq \log n$ when $k \leq \alpha n$.

□

Next we show the correctness of the construction. We first establish a series of lemmas.

Lemma 3.3.14. *For any* $i \leq L - 1$*, if* $v[i]$ *is correctly recovered, then in the* i*-th level the number of bad pairs in* w *is at most* $2k$*.*

Proof. We prove by contradiction.

Suppose there are more than $2k$ bad pairs in w. By Lemma 3.3.3, there is a self-matching having at least k bad pairs. By picking all the wrong pairs in this matching, we get a completely wrong self-matching $\tilde{w}$ having size at least $k = \varepsilon n$. This is a contradiction to the fact that $h_1, \ldots, h_{l_i}$ is a sequence of ε-self-matching hash functions.

□

Next we show that w is large enough so that in each level Bob can recover many blocks of x correctly.

Lemma 3.3.15. *For any $i \leq L-1$, in the i-th level, we have $|w| \geq l_i - k$.*

Proof. The k edits which the adversary makes on x can change at most k blocks of x'. The remaining unchanged blocks induce a matching between x' and y of size at least $l_i - k$. Since w is the maximum matching between x' and y, $|w| \geq l_i - k$.

□

Lemma 3.3.16. *Bob computes x correctly.*

Proof. Alice can do every step in her algorithm correctly due to Theorem 3.3.5 and 3.2.12 since she only constructs ε-self-matching hash functions, doing evaluation of these functions and computing redundancies of some sequences.

So the remaining is to show that Bob can compute x correctly once he receives Alice's message.

We first show that for every level i, Bob can recover $v[i]$ correctly, by indcution.

For the first level, Bob can do it because $v[1]$ is sent directly to him from Alice.

For level $i = 2, \ldots, L-1$, assume Bob gets $v[i-1]$ correctly. By Lemma 3.3.14, the matching w has at most $2k$ bad pairs. By Lemma 3.3.15, $|w| \geq l_i - k$. Thus $|w|$ gives at least $l_i - k - 2k = l_i - 3k$ correctly matched pairs of blocks. So according to w, Bob can recover at least $l_i - 3k$ blocks of x correctly. Thus in the i-th level, there are at most $3k \times 2 = 6k$ wrong blocks in $\tilde{x}$. So $h_1(\tilde{x}_1) \circ \ldots \circ h_{l_i}(\tilde{x}_{l_i}) \circ z[i]$ is a word in

$(\{0,1\}^{b^*})^{l_i+\Theta(k)}$ having distance at most $6k$ to a codeword of an $(l_i+\Theta(k), l_i, \Theta(k))$-code. Let the distance of the code $\Theta(k)$ be at least $14k$ s.t. by Theorem 3.2.12 the decoding algorithm can compute $v[i]$ correctly.

Note that for the last level, there are at least $n-3k$ correctly matched pairs of blocks. So there are at most $3k$ wrong blocks in $\tilde{x}$. The redundancy length $|z_{\mathsf{final}}| = \Theta(k)$. So $\tilde{x} \circ z_{\mathsf{final}}$ has hamming distance at most $3k$ from a codeword $x \circ z_{\mathsf{final}}$ of an $(n+\Theta(k), n, \Theta(k))$-code with distance at least $8k$. Thus the decoding algorithm of Theorem 3.2.12 can compute the message x correctly.

□

Theorem 3.3.17. *There exists a deterministic protocol for document exchange, having communication complexity (redundancy) $O(k \log^2 \frac{n}{k})$, time complexity $\mathsf{poly}(n)$, where n is the input size and k is the edit distance upper bound.*

Proof. Construction 3.3.11 gives the deterministic protocol for document exchange. The communication complexity is $O(k \log^2 \frac{n}{k})$ by Lemma 3.3.13. The time complexity is $\mathsf{poly}(n)$ by Lemma 3.3.12. The correctness is proved by Lemma 3.3.16.

□

3.4 Explicit binary ECC for edit errors

In this section we'll show how to use the document exchange protocol to construct ECC for edit errors.

3.4.1 Constructing binary ECC using redundancies

Lemma 3.4.1. *For every $n, r, k \in \mathbb{N}$, every $x \in \{0,1\}^n, z \in \{0,1\}^r$, if C is a binary code with message length r, codeword length n_C, that can correct up to $4k$ edit errors, then for every $y \in \{0,1\}^*$ s.t. $\mathsf{ED}(y, x \circ C(z)) \leq k$, one can get z from y using the decoding algorithm of C.*

Proof. We can run the decoding of C on $y[n+1-k, |y|]$, where $|y| \leq n + n_C + k$. Since there are at most k edit operations, $\mathsf{LCS}(y[n+1-k, |y|], C(z)) \geq n_C - k$. So

$$\begin{aligned}
\mathsf{ED}(y[n+1-k, |y|], C(z)) &= |y[n+1-k, |y|]| + n_C - 2\mathsf{LCS}(y[n+1-k, |y|], C(z)) \\
&\leq |y[n+1-k, |y|]| + n_C - 2(n_C - k) \\
&= |y| - (n+1-k) + 1 + n_C - 2(n_C - k) \\
&\leq (n + n_C + k) - (n+1-k) + 1 + n_C - 2(n_C - k) \\
&\leq 4k.
\end{aligned} \tag{3.1}$$

Thus the decoding can output the correct z.

□

Theorem 3.4.2. *If there exists an explicit document exchange protocol with communication complexity $r(n,k)$, where $n \in \mathbb{N}$ is the input size and $k \in \mathbb{N}$ is the upper bound on the edit distance, then there exists an explicit family of binary ECCs with codeword length $n_C = n + O(r(n,k))$, message length n, that can correct up to k edit errors.*

Proof. The encoding algorithm is as follows: for message $x \in \{0,1\}^n$, we first

compute the redundancy z for x and $3k$ edit errors using the document exchange protocol. Then we encode the redundancy z to be $\tilde{c}$ using the binary ECC from Theorem 3.2.13 which can correct α fraction of errors with constant rate. Here α is a constant such that $\alpha|\tilde{c}| \geq 4k$. Assume the length of the code is $n_0 = O(r)$. The final codeword c is the concatenation of the original message x and the encoded redundancy $\tilde{c}$. That is, $c = x \circ \tilde{c}$ and $|c| = n + n_0$.

By Lemma 3.4.1, we can get the redundancy z from the corrupted codeword c'. Note that there are at most k edit errors, $\mathsf{ED}(c'[1, n+k], x) = n + k + n - 2\mathsf{LCS}(c'[1, n+k], x) \leq n + k + n - 2(n-k) \leq 3k$. Here $\mathsf{LCS}(c'[1, n+k], x) \geq n - k$ is because $c'[1, n+k]$ contains a subsequence of x which are the symbols that are not deleted. There are at most k deletions so the subsequence has length at least $n - k$.

Finally we run the document exchange protocol to compute the original message x, where Bob's string is $c'[1, n+k]$ and the redundancy from Alice is z.

□

3.4.2 Binary ECC for edit errors with almost optimal parameters

A direct corollary of Theorem 3.4.2 is the following binary ECC.

Theorem 3.4.3. *For any $n, k \in \mathsf{N}$ with $k \leq n/4$, there exists an explicit binary error correcting code with message length n, codeword length $n + O(k \log^2 \frac{n}{k})$ that can correct up to k edit errors.*

Proof. It follows directly from Theorem 3.4.2 and 3.3.17.

□

Corollary 3.4.4. *There exists a constant $0 < \alpha < 1$ such that for any $0 < \varepsilon \leq \alpha$ there exists an explicit family of binary error correcting codes with codeword length n and message length m, that can correct up to $k = \varepsilon n$ edit errors with rate $m/n = 1 - O(\varepsilon \log^2 \frac{1}{\varepsilon})$.*

3.5 Deterministic document exchange protocol for block edit errors

Definition 3.5.1 (Collision free hash functions)**.** *Given $n, p, q \in \mathbb{N}, p \leq n$ and a string $x \in \{0,1\}^n$, we say a hash function $h : \{0,1\}^p \to \{0,1\}^q$ is collision free (for x), if for every $i, j \in [n - p + 1]$, $h(x[i, i + p)) = h(x[j, j + p))$ if and only if $x[i, i + p) = x[j, j + p)$.*

Theorem 3.5.2. *There exists an algorithm which, on input $n, p, q \in \mathbb{N}, p \leq n, q = c_0 \log n$ for large enough constant c_0, $x \in \{0,1\}^n$, outputs a description of a hash function $h : \{0,1\}^p \to \{0,1\}^q$ that is collision free for x, in time $\mathsf{poly}(n)$, where the description length is $O(\log n)$.*

Also there is an algorithm which, given the description of h and any $u \in \{0,1\}^p$, can output $h(u)$ in time $\mathsf{poly}(n)$.

Proof. Let $\varepsilon = 1/\mathsf{poly}(n)$ be small enough. Let $g : \{0,1\}^d \to (\{0,1\}^q)^{\{0,1\}^p}$ be an ε-almost $2q$-wise independence generator from Theorem 3.2.5 with $d = O(\log \frac{2q \log(2^p q)}{\varepsilon})$. Here g outputs $q2^p$ bits and we view the output as an array indexed by elements in $\{0,1\}^p$, where each entry is in $\{0,1\}^q$.

To construct h, we try every seed $v \in \{0,1\}^d$. Let $h(\cdot) = g(v)[\cdot]$. This means that, for every $u \in \{0,1\}^p, h(u)$ is the value of the entry indexed by u in $g(v)$. For

any $i, j \in [n-p+1]$, we check whether $h(x[i, i+p)) = h(x[j, j+p))$ if and only if $x[i, i+p) = x[j, j+p)$. If this is the case then the algorithm returns h. The description of h is the corresponding seed v.

Now we show that we can indeed find such a v by exhaustive search. If we let v be chosen uniformly randomly, then by a union bound, the probability that there exists $i, j \in [n-p+1]$ s.t. $h(x[i, i+p)) = h(x[j, j+p))$ but $x[i, i+p) \neq x[j, j+p)$ is at most $1/\mathsf{poly}(n) \cdot n^2 = 1/\mathsf{poly}(n)$. Thus there exists a v s.t. the corresponding h is collision free.

The exhaustive search is in polynomial time because the seed length is $d = O(\log n)$. The evaluation of h is in polynomial time by Theorem 3.2.5. Thus the overall running time of our algorithm is a polynomial in n. □

Definition 3.5.3 (Matching)**.** *Given $n, n', p, q \in \mathbb{N}, p \leq n, p \leq n'$, a function $h : \{0,1\}^p \to \{0,1\}^q$ and two strings $x \in \{0,1\}^n, y \in \{0,1\}^{n'}$, a matching (may not be monotone) between x and y under h is a sequence of matches (pairs of indices) $w = ((i_1, j_1), \ldots (i_{|w|}, j_{|w|}))$ s.t.*

- *for every $k \in [|w|]$,*
  - *$i_k = 1 + pl_k \in [n]$ for some l_k,*
  - *$j_k \in [n']$,*
  - *$h(x[i_k, i_k + p)) = h(y[j_k, j_k + p))$,*
- *$i_1, \ldots, i_{|w|}$ are distinct.*

A non-overlapping matching is a matching with one more restriction.

- *Intervals $[j_k, j_k + p), k \in [|w|]$, are disjoint.*

When considering overlaps, the matching has overlapping degree d, if each bit of y appears in at most d matched pairs for some small number d.

For a match (i,j), it matches two intervals, one from x, the other from y. When we say the y's interval (of the match (i,j)), we mean $[j,j+p)$, and similarly the x's interval is $[i,i+p)$. A match (i,j) in a matching is called a wrong match (or wrong pair) if $x[i,i+p) \neq y[j,j+p)$. Otherwise it is called a correct match (or correct pair). A pair of indices (i,j) is called a potential match between x and y if $h(x[i,i+p)) = h(y[j,j+p))$. It may be wrong because $x[i,i+p)$ may not be $y[j,j+p)$. When x,y are clear from the context we simply say (i,j) is a potential match.

To compute a monotone non-overlapping matching we can use the dynamic programming method. But our matching is not necessarily monotone. So this raises the question of how hard this problem is.

It seems difficult to find a polynomial algorithm which can exactly compute it. So instead we use constant approximation techniques. There're two difficulties at the first thought. One is that if we compute the non-overlapping matching over the entire strings, then a constant approximation is too bad since there will be $O(n)$ unmatched blocks. So for each level, we restrict our attention to blocks that are uncovered and wrongly recovered (but discovered by us). The other problem is that we need the approximation rate to be a large enough constant. To achieve this goal, we actually computing matchings with constant degree.

We start from a $1/3$-approximation algorithm, which is greedy.

Construction 3.5.4. *Given $n,n',p,q \in \mathbb{N}, p \leq n, p \leq n'$, a polynomial time computable function $h : \{0,1\}^p \to \{0,1\}^q$ and two strings $x \in \{0,1\}^n, y \in$*

$\{0,1\}^{n'}$, *we have the following* $1/3$*-approximation algorithm for computing the non-overlapping matching.*

1. *Let the sequence of matches* w *be empty;*
2. *Find* $i = 1 + pl \in [n]$ *and* $j \in [n']$, *where* $l \in \mathbb{N}$, *s.t.*
   - $h(x[i, i+p)) = h([j, j+p))$,
   - i *is not in any match (as the first entry) of the current* w,
   - $[j, j+p)$ *does not overlap with any* $[j', j'+p)$ *for any* j' *as the second entry in any matches of the current* w;
3. *If there is such a pair of indices* i, j, *then add the match* (i, j) *to* w *and go to step 2; Otherwise, output* w *and stop.*

Lemma 3.5.5. *Construction 3.5.4 gives a* $1/3$*-approximation algorithm for computing the non-overlapping matching.*

Proof. Suppose w^* is the maximum non-overlapping matching between x, y under h.

Every time the greedy algorithm adds a match (i, j) to w, we may delete at most 3 matches in w^*. They may be the match which includes $[i, i+p)$, or the matches whose intervals of y overlap with $[j, j+p)$.

Note that in the first case, there can be at most 1 match of w^* deleted since by definition of matching, $[i, i+p)$ can only be the x's interval for at most l match of w^*. For the second case, note that since w^* is non-overlapping, there are at most two y's intervals, of matches in w^*, overlapping with $[j, j+p)$.

If $|w| < 1/3|w^*|$, then we can delete less than $|w^*|$ matches in w^*.

We claim that the matches left can be selected by the greedy algorithm. Suppose one remaining match is (i, j). Note that $[i, i+p)$ is not in any match of w. Since if it is, then this match should have been deleted. Also note that $[j, j+p)$ does not overlap with any intervals in matches of w. Since if it does, then it also should have been deleted.

As a result, if $|w| < 1/3|w^*|$, our greedy algorithm will not stop. Also note that every time the algorithm conducts step 2 and 3 it will either increase the current matching size by 1, or stop, and the matching size is $O(n/p)$. So our greedy algorithm will halt in polynomial time.

□

Next we give an explicit algorithm which computes a even larger matching (better approximation), but it allows overlaps.

Construction 3.5.6. *Given $n, n', p, q \in \mathbb{N}, p \leq n, p \leq n'$, a (polynomial time computable) function $h : \{0,1\}^p \to \{0,1\}^q$ and two strings $x \in \{0,1\}^n, y \in \{0,1\}^{n'}$, we have the following algorithm.*

1. *Let the matching w be empty, set $S = \{i = 1 + pl \mid l \in \mathbb{N}, i \in [n]\}$, integer $c = 0$;*
2. *Conduct Construction 3.5.4 to compute a matching w' between x_S and y under h. Here x_S is the projection of x on intervals in set S;*
3. *Let $w = w \cup w'$;*
4. *Let $S = S \setminus \{u \mid \exists (u, v) \in w\}$;*
5. *$c = c + 1$;*

6. *If $c \geq 3$, output w; Otherwise go to step 2.*

Note that Construction 3.5.6 is in polynomial time since it simply conducts Construction 3.5.4 for 3 times and after each conduction it removes matched blocks of x and only considers the remaining blocks in the next iteration. So we only need to show its correctness.

Lemma 3.5.7. *Construction 3.5.6 computes a degree 3 overlapping matching w between x and y under h, such that $|w| \geq 2/3|w^*|$, where w^* is the maximum non-overlapping matching between x and y under h.*

Proof. Let $w_i, i = 1,2,3$ be the matching the algorithm computes after round i. Also let $S_i, i = 1,2,3$ be the set S after the ith round.

By Lemma 3.5.5, $|w_1| \geq 1/3|w^*|$. The number of unmatched blocks is $\bar{n} - |w_1| \leq \bar{n} - 1/3|w^*|$, where $\bar{n} = \lfloor n/p \rfloor$ is the total number of blocks of x.

The maximum matching between x_{S_1} and y is at least $|w^*| - |w_1|$. This is because that, each of the matched blocks of x by w_1, should be among the x's blocks in the matches of w^*. There are at most $|w_1|$ of them. So there are still $|w^*| - |w_1|$ remaining matches in w^* which corresponds to blocks in x_{S_1}.

Again by Lemma 3.5.5, for $i \geq 2$, at least $1/3(|w^*| - |w_{i-1}|)$ blocks of $x_{S_{i-1}}$ will be matched in the ith round.

Thus

$$
\begin{aligned}
|w_i| &\geq |w_{i-1}| + 1/3(|w^*| - |w_{i-1}|) && (3.2)\\
&= 1/3|w^*| + 2/3|w_{i-1}| && (3.3)\\
&\geq (1-(2/3)^{i-1})|w^*| + (2/3)^{i-1}|w_1| && (3.4)\\
&\geq (1-(2/3)^{i-1})|w^*| + (1/3)(2/3)^{i-1}|w^*| && (3.5)\\
&= (1-(2/3)^{i})|w^*|. && (3.6)
\end{aligned}
$$

Inequality 3.2 is due to Lemma 3.5.5 as explained above. Equality 3.3 is due to a direct computation. 3.4 is by recursively applying 3.2 and 3.3 from $i-1$ to 2. 3.5 is because $|w_1| \geq 1/3|w^*|$.

As a result, $|w_3| \geq 19/27|w^*| \geq 2/3|w^*|$.

Note that we apply Construction 3.5.4 for 3 times, where in each time, it gives a non-overlapping matching. So each entry of y is in at most one of the matches in that round. So finally we get a degree 3 overlapping matching. □

We now give the following document exchange protocol.

Construction 3.5.8. *The protocol works for every input length $n \in \mathbb{N}$, every (k_1, t) block-insertions/deletions k_2 block-transpositions, $k_1, k_2 \leq \alpha n/\log n, t \leq \beta n$, for some constant α, β. (If k_1 or $k_2 > \alpha n/\log n$, or $t > \beta n$, we simply let Alice send her input string.) Let $k = k_1 + k_2$.*

Both Alice's and Bob's algorithms have $L = O(\log \frac{n}{k \log n + t})$ levels.

For every $i \in [L]$, in the i-th level,

- *Let the block size be* $b_i = \frac{n}{18 \cdot 2^i (k + \frac{t}{\log n})}$*, i.e., in each level, divide every block of* x *in the previous level evenly into two blocks. We choose* L *properly s.t.* $b_L = O(\log n)$*;*

- *The number of blocks* $l_i = n / b_i$*;*

Alice: On input $x \in \{0,1\}^n$*,*

1. *For the* i*-th level,*

   1.1. *Construct a hash function* $h_i : \{0,1\}^{b_i} \to \{0,1\}^{b^* = \Theta(\log n)}$ *for* x *by Theorem 3.5.2.*

   1.2. *Compute the sequence of hash values i.e.* $v[i] = (h_i(x[1, 1+b_i)), h_i(x[1+b_i, 1+2b_i)), \ldots, h_i(x[1+(l_i-1)b_i, l_i b_i)))$*;*

   1.3. *Compute the redundancy* $z[i] \in (\{0,1\}^{b^*})^{\Theta((k+\frac{t}{\log n})i)}$ *for* $v[i]$ *by Theorem 3.2.12, where the code has distance at least* $180(k + \frac{t}{\log n})i$*;*

2. *Compute the redundancy* $z_{\text{final}} \in (\{0,1\}^{b_L})^{\Theta((k+\frac{t}{\log n}) \log L)}$ *for the blocks of the* L*-th level by Theorem 3.2.12, where the code has distance at least* $90(k + \frac{t}{\log n})L$*;*

3. *Send* $h = (h_1, \ldots, h_L)$*,* $z = (z[1], z[2], \ldots, z[L])$*,* $v[1]$*,* z_{final} *to Bob.*

Bob: On input $y \in \{0,1\}^{O(n)}$ *and received* h, z*,* $v[1]$*,* z_{final}*,*

1. *Create* $\tilde{x} \in \{0,1,*\}^n$ *(i.e. Bob's current version of Alice's* x*), initiating it to be* $(*, *, \ldots, *)$*;*

2. *For the* i*-th level where* $1 \le i \le L-1$*,*

2.1. *Apply the decoding of Theorem 3.2.12 on* $h_i(\tilde{x}'[1, 1+b_i)), h_i(\tilde{x}'[1+b_i, 1+2b_i)), \ldots, h_i(\tilde{x}'[1+(l_i-1)b_i, l_i b_i)), z[i]$ *to get the sequence of hash values* $v[i]$*. Note that* $v[1]$ *is received directly, thus Bob does not need to compute it;*

2.2. *Let* $S = \{j \in [n] \mid h_i(\tilde{x}[1+(j-1)b_i, 1+jb_i)) \neq v[i][j]$ *or* $x[1+(j-1)b_i, 1+jb_i) = (*, \ldots, *)\}$*;*

2.3. *Compute the matching* $w_i = ((p_1, p'_1), \ldots, (p_{|w|}, p'_{|w|})) \in ([l_i] \times [|y|])^{|w_i|}$ *between* x_S *and* y *under* h_i*, using* $v[i]$*, by Lemma 3.5.6;*

2.4. *Evaluate* $\tilde{x}$ *according to the matching, i.e. let* $\tilde{x}[p_j, p_j + b_i) = y[p'_j, p'_j + b_i)$*, where* $p_j, p'_j \in w_i, j \in [|w_i|]$*;*

3. *In the* L*'th level, apply the decoding of Theorem 3.2.12 on the blocks of* $\tilde{x}$ *and* z_{final} *to get* x*;*

4. *Return* x*.*

Lemma 3.5.9. *For every* i*, the maximum non-overlapping matching between* x_S *and* y *under* h_i *has size at least* $|S| - (2k_1 + 3k_2 + t/\log n)$*.*

Proof. Note that block-insertions do not delete bits. One block insertion can corrupt at most one block. For block-deletions, assume that the j-th block-deletion delete t_j bits. This can corrupt (delete a block totally or delete part of a block) at most $\lceil t_j/b_i \rceil + 1$ blocks. So the total number of corrupted blocks is at most $\sum_{j=1}^{k_1}(\lceil t_j/b_i \rceil + 1) \leq 2k_1 + t/b_i \leq 2k_1 + t/\log n$.

On the other hand, k_2 block-transpositions can corrupt at most $3k_2$ blocks, because one block-transposition can only corrupt the two blocks at the end of the transposed

substring and another block which contains the position that is the destination of the transposition.

As a result, the total number of corrupted blocks is at most $2k_1 + 3k_2 + t/\log n$. After corruption, uncorrupted blocks can be matched to its corresponding blocks (before corruption) in x. So there exists a matching between x_S and y under h_i having size at least $|S| - (2k_1 + 3k_2 + t/\log n)$.

□

Lemma 3.5.10. *For every i, if $v[i]$ is correctly computed by Bob, then $|w_i| \geq 2/3(|S| - (2k_1 + 3k_2 + t/\log n))$.*

Proof. By Lemma 3.5.9, the maximum non-overlapping matching between x_S and y under h_i has size at least $|S| - (2k_1 + 3k_2 + t/\log n)$. By Lemma 3.5.7, $|w_i| \geq 2/3(|S| - (2k_1 + 3k_2 + t/\log n))$.

□

Lemma 3.5.11. *For every i, if $v[1], \ldots, v[i]$ are correctly recovered, then in the i-th level the number of wrongly recovered blocks of x is at most $3i(2k_1 + 3k_2 + \frac{t}{\log n})$.*

Proof. Consider the matching w^* corresponding to the current recovering of x after i levels, i.e., this matching is generated at level 1 and adjusted level by level. In level j, we first use hash values to test every block to see if it is correctly recovered. For wrongly recovered blocks we delete their corresponding matches. Then for remaining wrongly recovered blocks and unrecovered blocks, we compute a matching w_j for them, and add all matches in w_j to w^*.

For $w_j, j \leq i$, after level i, the number of wrongly recovered blocks in level i caused by (the remaining part of) w_j is at most $3(2k_1 + 3k_2 + \frac{t}{\log n})$.

This is because in w_j is constructed by Construction 3.5.6, which is a union of 3 matchings. Each matching of them is non-overlapping. We only need to show that w_j, after eliminating detected wrong pairs in these i levels, contains at most $2k_1 + 3k_2 + \frac{t}{\log n}$ wrong matches between x's and y's blocks in the i-th level. To see this, first note that these matches' y intervals are only from blocks which are modified from x's blocks or newly inserted. For each block-insertion of t_j bits, it can contribute at most $\lceil t_j/b_i \rceil + 1$ wrong matches. Each block-deletion can contribute at most 2 wrong matches. So totally block insertions/deletions can cause $\sum_{j=1}^{k_1}(\lceil t_j/b_i \rceil + 1) \leq 2k_1 + t/b_i$ wrong matches. On the other hand, k_2 block-transpositions can contribute at most $3k_2$ wrong matches, because 1 block-transposition can only cause 1 wrong match when deleting the block and inserting the block to its destination may contribute 2 wrong matches. Hence the total number wrong matches is at most $2k_1 + 3k_2 + t/b_i$.

Since there are i matchings $w_1, \ldots, w_i$, each containing 3 non-overlapping matchings, the number of wrongly recovered blocks remaining in w^* is at most $3i(2k_1 + 3k_2 + \frac{t}{\log n})$.

□

Lemma 3.5.12. *For every i, if $v[1], \ldots, v[i]$ are correctly recovered, then in level i, the number of unrecovered blocks is at most $36i(k + \frac{t}{\log n})$.*

Proof. We use induction.

For the base case $i = 1$, all blocks of x are unknown to Bob. So the number is at most $l_1 = 18 \cdot 2^1(k + t/\log n) = 36(k + t/\log n)$.

For the induction case, assume the number of unrecovered blocks is at most $36j(k + t/\log n)$, for all $j \leq i$. By Lemma 3.5.11, for level i (after the matching is

computed), the number of wrongly recovered blocks of x is at most

$$3i(2k_1 + 3k_2 + \frac{t}{\log n}).$$

So at level $i+1$, the number of wrongly recovered blocks is at most doubled, i.e.

$$\begin{aligned}
&|\{j \in [n] \mid h_{i+1}(\tilde{x}[1+(j-1)b_{i+1}, 1+jb_{i+1})) \neq v[i][j]\}| \\
\leq &6i(2k_1 + 3k_2 + \frac{t}{\log n}) \\
\leq &18i(k + t/\log n).
\end{aligned}$$

Since Bob has the correct $v[i+1]$, he can detect at most all the wrong blocks. So $|S| \leq (72+18)i(k+t/\log n) = 90i(k+t/\log n)$.

By Lemma 3.5.10, the number of unrecovered blocks is at most $|S| - |w_i| \leq 1/3|S| + (2k_1 + 3k_2 + t/\log n) \leq 30(i+1)(k+t/\log n)$.

□

Lemma 3.5.13. *Bob can recover x correctly.*

Proof. We use induction to show that for every $i \in [L]$, $v[i]$ can be computed correctly by Bob.

For the first level, $v[1]$ is directly received from Alice.

Assume $v[1], \ldots, v[i-1]$ can be computed correctly. By Lemma 3.5.12, the number of unrecovered blocks after level $i-1$ is at most $36(i-1)(k+t/\log n)$. By Lemma 3.5.11, the number of wrongly recovered blocks is at most $9(i-1)(k + t/\log n)$. So the total number of wrongly recovered and unrecovered blocks is at

most

$$(36(i-1)(k+t/\log n)+9(i-1)(k+t/\log n))$$

$$\leq 90(i-1)(k+t/\log n)$$

$$< 90i(k+t/\log n).$$

Note that with the redundancy $z[i]$, its corresponding code has distance at least $180(k+t/b_i)i$. So Bob can recover $v[i]$ correctly by Theorem 3.2.12.

As a result, at level L. By Lemma 3.5.11, the number of wrongly recovered blocks is at most $3L(2k_1+3k_2+\frac{t}{b_L})$. By Lemma 3.5.12 the number of unrecovered blocks, is at most $36L(k+t/\log n)$. So the total number of wrongly recovered and unrecovered blocks is at most $45L(k+t/\log n)$. Note that the code distance corresponding to the redundancy z_{final} is at least $90(k+t/b_L)L$. So all blocks of x can be recovered correctly by using the decoding from Theorem 3.2.12.

□

Lemma 3.5.14. *The communication complexity is* $O((k\log n+t)\log^2\frac{n}{k\log n+t})$.

Proof. For the i-th level of Alice, $|z[i]| = \Theta(k+\frac{t}{\log n})ib^* = \Theta((k\log n+t)i)$. So

$$|z| = \sum_{i=1}^{L}|z[i]| = \sum_{i=1}^{L} O\left((k\log n+t)i\right) = O(k\log n+t)L^2.$$

Also $|z_{\text{final}}| = O(k+\frac{t}{b_L})\cdot L\cdot O(\log n) = O\left((k\log n+t)L\right)$ by Theorem 3.2.12.

For every $i\in[L]$, $|h_i| = O(\log n)$ by Theorem 3.5.2. So $|h| = O(\log n)L$.

The length of $v[1]$ is $l_1 O(\log n) = \frac{n}{b_1}O(\log n) = O(k+\frac{t}{\log n})\cdot O(\log n) =$

$O(k \log n + t)$.

Since $L = \log \frac{n}{k \log n + t}$, the overall communication complexity is

$$O\left((k \log n + t) \log^2 \frac{n}{k \log n + t} \right).$$

□

Lemma 3.5.15. *Both Alice and Bob's algorithms are in polynomial time.*

Proof. For Alice's algorithm, let's consider the i-th level. Constructing h_i and evaluating h_i takes polynomial time by Theorem 3.5.2. Computing the redundancy $z[i]$ takes polynomial time by Theorem 3.2.12. So the overall running time is polynomial.

For Bob's algorithm, we still consider the i-th level. By Theorem 3.2.12, getting $v[i]$ takes polynomial time. It takes linear time to visit every block and check if their hash value is equal to the corresponding entry of $v[i]$. By Lemma 3.5.7, computing the maximum matching takes polynomial time. So the overall running time is also polynomial.

□

Theorem 3.5.16. *There exists an explicit binary document exchange protocol, having communication complexity* $O((k \log n + t) \log^2 \frac{n}{k \log n + t})$, *time complexity* $\mathsf{poly}(n)$, *where* n *is the input size and* $k = k_1 + k_2$, *for* (k_1, t) *block-insertions/deletions and* k_2 *block-transpositions,* $k_1, k_2 \leq \alpha n / \log n, t \leq \beta n$, *for some constant* α, β.

Proof. It follows from Construction 3.5.8, Lemma 3.5.13, Lemma 3.5.14 and Lemma 3.5.15.

□

3.6 Binary codes for block edit errors

3.6.1 Encoding and decoding algorithm

Given the document exchange protocol for block edit operations, we can now construct codes capable of correcting (k_1, t) block insertions/deletions, and k_2 block transpositions, where $k_1 + k_2 = k$, and $t \leq \alpha n$ for some constant α. The encoding and decoding algorithms are as follows:

Construction 3.6.1. *Encoding Algorithm*

Let $\ell_{\mathsf{buf}} = 2\log n$ *and* $\mathsf{buf} = 0^{\ell_{\mathsf{buf}}-1} \circ 1$.

Input*:* msg *of length* n.

Ingredients*:*

- *A pseudorandom generator* $\mathsf{PRG} : \{0,1\}^{O(\log n)} \to \{0,1\}^n$*, from Theorem 3.6.3, s.t. there exists at least one seed* r *for which* $msg \oplus \mathsf{PRG}(r)$ *doesn't contain* buf *as a substring and has B-distinctness.*
- *An error correcting code* $\mathcal{C}_1$ *from Theorem 3.2.13 which is capable of correcting* $O(k\log n + t)$ *edit errors, as well as* k_2 *block transpositions. Denote the encoding map of* $\mathcal{C}_1$ *as* $Enc_1 : \{0,1\}^{\mathsf{ml}_1 = O(k\log^2 n + t)} \to \{0,1\}^{\mathsf{cl}_1 = O(k\log^2 n + t)}$ *and the decoding map as* $Dec_1 : \{0,1\}^{\mathsf{cl}_1'} \to \{0,1\}^{\mathsf{ml}_1}$.

Operations*:*

1. *Find a seed* r *of* PRG *s.t.* $\mathsf{msg} \oplus \mathsf{PRG}(r)$ *does not contain* buf *as a substring and satisfies B-distinctness. Let* $\mathsf{msg}_P = \mathsf{msg} \oplus \mathsf{PRG}(r)$.

2. *Compute the sketch* sk_m *for* msg_P *for* $\Omega(k)$ *block insertions/deletions and* $\Omega(k)$ *block transpositions, where the number of bits inserted and deleted is* $\Omega(k \log n + t)$ *in total.*

3. *Let* $\mathsf{sk} = \mathsf{sk}_m \circ r$*, and encode* sk *with* $\mathcal{C}_1$*. Let the codeword be* $c_1 = Enc_1(\mathsf{sk})$*.*

4. *Divide* c_1 *into blocks of length* $\log n$*. Denote these blocks as* $c_1^{(1)}, c_1^{(2)}, \ldots, c_1^{(M)}$ *where* M *is the number of blocks.*

5. *Insert* buf *to the beginning of each block* $c_1^{(i)}, 1 \leq i \leq M$*.*

6. *Let* $c = (\mathsf{msg} \oplus \mathsf{PRG}(r)) \circ \mathsf{buf} \circ c_1^{(1)} \circ \mathsf{buf} \circ c_1^{(2)} \cdots \circ \mathsf{buf} \circ c_1^{(M)}$*.*

Output*: c.*

The construction of PRG is left to subsection 3.6.2. We call the concatenation $\mathsf{buf} \circ c_1^{(1)} \circ \mathsf{buf} \circ c_1^{(2)} \cdots \circ \mathsf{buf} \circ c_1^{(M)}$ as the sketch part and $\mathsf{msg}_P = \mathsf{msg} \oplus \mathsf{PRG}(r)$ as the message part. Now we give the corresponding decoding algorithm.

Construction 3.6.2. *Decoding Algorithm*

Input*: the received codeword* c'*.*

Operations*:*

1. *Find out all substrings* buf *in* c'*. Number these buffers as* $\mathsf{buf}_1, \ldots, \mathsf{buf}_{M'}$*.*

2. *Pick the* $\log n$ *bits after* buf_j *as block* $c_1'^{(j)}, 1 \leq j \leq M'$*. Then remove all the buffers* buf_j *and* $c_1'^{(j)}$*,* $1 \leq j \leq M'$ *from* c'*. The rest of* c' *is regarded as the message part* $\mathsf{msg}_P^{'}$*.*

3. *Let* $c_1' = c_1'^{(1)} \circ c_1'^{(2)} \circ \cdots \circ c_1'^{(M')}$*. Decode* c_1' *with the decoding algorithm* Dec_1 *for* $\mathcal{C}_1$ *and get* $\mathsf{sk} = Dec_1(c_1')$*.*

4. *Get* sk_m *and* r *from* sk.

5. *Use* sk_m *and* $\mathsf{msg}_P^{'}$ *to recover* msg_P.

6. *Compute* $\mathsf{msg} = \mathsf{msg}_P \oplus \mathsf{PRG}(r)$.

Output*:* msg.

3.6.2 Analysis

In this subsection we'll give the construction of PRG and prove the correctness of the algorithms.

Building blocks: PRG

We recall the following pseudorandom generator.

Theorem 3.6.3. *For every $n \in \mathbb{N}$, there exists an explicit PRG $g : \{0,1\}^{\ell=O(\log n)} \to \{0,1\}^n$ s.t. for every $x \in \{0,1\}^n$, with probability $1 - 1/\mathsf{poly}(n)$, $g(U_\ell) + x$ satisfies B-distinctness.*

Proof. Let g be the ε-almost κ-wise independence generator from Theorem 3.2.5, where $\varepsilon = 1/\mathsf{poly}(n)$, $\kappa = 2B$, seed length $\ell = O(\log \frac{\kappa \log n}{\varepsilon}) = O(\log n)$.

Given a fixed x, $g(n, U_\ell) + x$ is ε-almost κ-wise independent. So for every pair of two intervals $u, v \in \{0,1\}^B$ of it,

$$\Pr[u = v] = \sum_{a \in \{0,1\}^B} \Pr[u = a, v = a] \leq \sum_{a \in \{0,1\}^B} (\frac{1}{2^{2B}} + \varepsilon) \leq \frac{1}{2^B} + 1/\mathsf{poly}(n) \leq \frac{1}{2^{B-1}},$$

where the first inequality is due to the definition of ε-almost κ-wise independence. The second inequality holds since $\varepsilon = 1/\mathsf{poly}(n)$ is small enough. The third inequality

is because ε is small enough and we can take the constant in $B = O(\log n)$ to be large enough. By a union bound over all $O(n^2)$ pair of u, v, it concludes that $g_3(n, U_\ell) + x \in \{0,1\}^n$ is B-distinct with probability $1 - 1/\mathsf{poly}(n)$ since B is large enough.

□

Theorem 3.6.4. *For every $n \in \mathbb{N}, x \in \{0,1\}^n$,there exists an explicit PRG $g : \{0,1\}^{\ell=O(\log n)} \to \{0,1\}^n$ s.t. for every $x \in \{0,1\}^n$, with probability $1 - 1/\mathsf{poly}(n)$, the following two conditions hold simultaneously.*

- buf *is not a substring of* $\mathsf{PRG}(U_\ell) \oplus x$.
- $\mathsf{PRG}(U_\ell) \oplus x$ *satisfies B-distinctness.*

Proof. Let $\kappa = \ell_{\mathsf{buf}}$ be the length of buf, $\varepsilon = 1/n^2$. From Theorem 3.2.5, there exists an explicit ε-almost κ-wise independence generator $g' : \{0,1\}^d \to \{0,1\}^n$, where $d = O(\log \frac{\kappa \log n}{\varepsilon}) = O(\log n)$. Then, for any $x \in \{0,1\}^n$,

$$\begin{aligned}
&\Pr_{r' \leftarrow \{0,1\}^d}[\mathsf{buf} \text{ is a substring of } g'(r') \oplus x] \\
\leq &\sum_{i \in [n - \ell_{\mathsf{buf}} + 1]} \Pr[\mathsf{buf} = (g'(r') \oplus x)[i, i + \ell_{\mathsf{buf}})] \\
\leq &n \left(1/2^{\ell_{\mathsf{buf}}} + 1/n^2\right) = 1/\mathsf{poly}(n).
\end{aligned}$$

Let g be the generator in Theorem 3.6.3 with seed length ℓ'. Let $\ell = \max(\ell', d)$, and construct $\mathsf{PRG}(r) = g(r_1) \oplus g'(r_2)$ where r_1, r_2 are disjoint substrings of r of length l' and l. Then by the union bound, the probability that at least one of the conditions fails is upper bounded by $1/\mathsf{poly}(n)$. □

Correctness of the construction

We show that a code $\mathcal{C}$ with encoding algorithm 3.6.1 and decoding algorithm 3.6.2 can correct (k_1, t)-block insertions/deletions and k_2 block transpositions.

First, we prove the sketch sk can be correctly recovered.

Lemma 3.6.5. *In the 4th step of decoding algorithm 3.6.2, the sketch* sk *is correctly recovered.*

Proof. We show that c_1' can be obtained by applying at most $12k \log n + t$ edit errors and k block transpositions over c_1.

Note that after inserting buffers to the blocks of c_1, the total number of appearance of the buffer in the sketch part is equal to the number of buffers inserted, because the buffer length is longer than the block length of c_1. Also note that concatenating the message part and sketch part will not insert any buffers because by the choice of r, $\mathsf{msg} \oplus \mathsf{PRG}(r)$ does not contain buf. As a result, if there are no errors, by the decoding algorithm we can get the correct c_1 and thus get the correct sk.

Next we consider the effects of block insertions/deletions and transpositions for the sketch part. Specifically, we consider how the sketch part changes after each of these operations.

- block insertion: Consider one block insertion of t_0 bits. We claim that after this operation, at most $\lceil t_0/(3 \log n) \rceil$ new blocks can be introduced to the sketch part, because to insert one new block to the sketch, we only need to insert a new buffer and attach the new block to it. We also note that this operation may delete one block by damaging a buffer, or replace one block by damaging the block right after the buffer.

So k_1 block insertions of t bits inserted can insert at most $k_1 + t/(3\log n)$ new blocks. It can also delete at most k_1 blocks, and replace at most k_1 blocks.

- block deletion: we first consider a block deletion of t_0 bits. After this operation, at most $\lceil t_0/(3\log n)\rceil$ blocks of the sketch part can be deleted, since there are at most $\lceil t_0/(3\log n)\rceil$ blocks in the deleted substring. The operation may also create one extra block, since the remaining bits may combine together to be a buffer. It may also replace an existing block, since the remaining bits may combine together to be a new block after an original buffer.

  So k_1 block deletions of t bits deleted can delete at most $k_1 + t/(3\log n)$ blocks. It can insert at most k_1 blocks. It can also replace k_1 blocks.

- block transposition: After one block transposition (i, j, l), at most 3 new blocks can be introduced to the sketch part, since a new block may be created at the original position i, and two new blocks may appear when inserting the block to the destination j. Also it may delete at most 3 blocks, since two buffers may be damaged when removing the transferred block, and one buffer can be damaged when inserting the transferred block. By a similar argument this operation can replace at most 3 blocks. Also, a block transposition can cause one block transposition for the sketch part.

  As a result, k_2 block transpositions can insert or delete at most $O(k_2)$ blocks and cause $O(k_2)$ block transpositions.

In summary, there are at most $O(k + t/\log n)$ block insertions/deletions and k_2 block transpositions on c_1. Note that $O(k + t/\log n)$ block insertions/deletions, each of length $O(\log n)$ bits can be regarded as $O(k\log n + t)$ edit errors. Since our code

$\mathcal{C}_1$ can correct $O(k \log n + t)$ edit errors and k_2 block transpositions, we can decode sk correctly.

□

Next, we show that the message output by the decoding algorithm is correct.

Lemma 3.6.6. *At the end of algorithm 3.6.2, the original message is correctly decoded.*

Proof. According to Lemma 3.6.5, we have correctly recovered sk. Thus we get sk_m and r correctly.

Note that if there are no errors, then by deleting the buffers and the blocks of c_1 appended to these buffers, the remaining string is exactly the original message part, since the original message part does not contain buf as substrings.

Now we consider the effects of block insertions/deletions and transpositions for the message part. Specifically, we consider how the message part changes after each of these operations.

- block insertion: First consider one block insertion of t_0 bits. It can insert at most t_0 symbols to the message part if it does not damaging any original buffers. If it damages buffers, it may insert $O(\log n)$ more bits to the message part. It can also cause at most one block deletion of $O(\log n)$ bits since the rightmost buffer it inserts may cause our algorithm to delete the $O(\log n)$ bits following that buffer.

- block deletion: Consider a block deletion of t_0 bits. It can delete at most t_0 blocks of the message part. If it damages buffers, it can cause at most one

block insertion of $O(\log n)$ bits, since the rightmost deleted buffer may cause our algorithm to regard the $O(\log n)$ bits following that buffer as part of the message part.

- block transposition: now we consider one block transposition. It may cause at most one block transposition of the message part. Also it may create at most 3 new buffers and thus delete $3\log n$ bits of the message part. Moreover, it may delete three buffers and thus insert $3\log n$ bits to the message part.

Thus (k_1, t)-block insertions/deletions can cause inserting/deleting at most $O(k_1)$ blocks of $O(t + k_1 \log n)$ bits. Also k_2 block transpositions can cause $O(k_2)$ block insertions/deletions of $O(k_2 \log n)$ bits in total and k_2 block transpositions.

In summary, there are at most $O(k)$ block insertions/deletions of $O(k \log n + t)$ bits in total and k_2 block transpositions. Since our sketch sk can be used to correct $(O(k), O(k \log n + t))$ block insertions/deletions and k block transpositions, we can get msg_P correctly. As a result we can compute $\mathsf{msg} = \mathsf{msg}_P \oplus \mathsf{PRG}(r)$ correctly.

□

We can directly use our document exchange protocol to get an ECC.

Theorem 3.6.7. *For every $n, k_1, k_2, t \in \mathbb{N}$ with $k = k_1 + k_2 < \alpha n / \log n, t \leq \beta n$, for some constant α, β, there exists an explicit binary error correcting code for (k_1, t)-block insertions/deletions and k_2 block transpositions, having message length n, codeword length $n + O((k \log n + t) \log^2 \frac{n}{k \log n + t})$.*

Proof. We construct the encoding as Algorithm 3.6.1 where the sketch in Stage 2 is computed by using Alice's algorithm (encoding) of the protocol of Theorem 3.5.16.

The decoding is as Algorithm 3.6.2, where its stage 5 is computed by using Bob's algorithm of the protocol of Theorem 3.5.16.

The correctness of the construction is similar to Lemma 3.6.5, 3.6.6, the (k_1, t)-block insertions/deletions and k_2 block transpositions causes $(k, O(k \log n + t))$-block insertions/deletions and transpositions on the message and sketch part. Hence, according to Theorem 3.5.16, a sketch of size $O((k \log n + t) \log^2 \frac{n}{k \log n + t})$ for the document exchange protocol is enough to correct the errors.

By Algorithm 3.6.1 and Theorem 3.2.13, the size of c_1 is $O((k \log n + t) \log^2 \frac{n}{k \log n + t})$. The total length of the buffer inserted is $O(\log n) \cdot |c_1| / \log n = O(|c_1|)$. Hence, the total length of the redundancy is $O((k \log n + t) \log^2 \frac{n}{k \log n + t})$. □

References

[NZ96] Noam Nisan and David Zuckerman. “Randomness is Linear in Space”. In: *Journal of Computer and System Sciences* 52.1 (1996), pp. 43–52.

[GUV09] Venkatesan Guruswami, Christopher Umans, and Salil Vadhan. “Unbalanced Expanders and Randomness Extractors from Parvaresh-Vardy Codes”. In: *Journal of the ACM* 56.4 (2009).

[CW79] J. L. Carter and M. N. Wegman. “Universal classes of hash functions”. In: *Journal of Computer and System Sciences* 18 (1979), pp. 143–154.

[Tre01] Luca Trevisan. “Extractors and Pseudorandom Generators”. In: *Journal of the ACM* (2001), pp. 860–879.

[Vio05b] Emanuele Viola. “The complexity of constructing pseudorandom generators from hard functions”. In: *computational complexity* 13.3-4 (2005), pp. 147–188.

[GVW15] Oded Goldreich, Emanuele Viola, and Avi Wigderson. “On Randomness Extraction in AC0”. In: *30th Conference on Computational Complexity (CCC 2015)*. Vol. 33. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. 2015, pp. 601–668.

[Can+00] Ran Canetti, Yevgeniy Dodis, Shai Halevi, Eyal Kushilevitz, and Amit Sahai. “Exposure-Resilient Functions and All-or-Nothing Transforms”. In: *Advances in Cryptology — EUROCRYPT 2000*. Ed. by Bart Preneel. Vol. 1807. Lecture Notes in Computer Science. Springer-Verlag, 2000, pp. 453–469.

[KZ07] Jesse Kamp and David Zuckerman. “Deterministic Extractors for Bit-Fixing Sources and Exposure-Resilient Cryptography”. In: *SIAM Journal on Computing* 36.5 (2007), pp. 1231–1247.

[Hås89] J. Håstad. “Almost Optimal Lower Bounds for Small Depth Circuits”. In: *Randomness and Computation*. Ed. by S. Micali. Vol. 5. JAI Press, 1989, pp. 143–170.

[BG13] Andrej Bogdanov and Siyao Guo. “Sparse extractor families for all the entropy”. In: *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*. ACM. 2013, pp. 553–560.

[DT09] Anindya De and Luca Trevisan. “Extractors Using Hardness Amplification”. In: *RANDOM 2009, 13th International Workshop on Randomization and Approximation Techniques in Computer Science*. 2009.

[Hås+93] 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* 28 (1993), pp. 12–24.

[IN96] Russell Impagliazzo and Moni Naor. “Efficient cryptographic schemes provably as secure as subset sum”. In: *Journal of cryptology* 9.4 (1996), pp. 199–216.

[Vio05a] Emanuele Viola. “On constructing parallel pseudorandom generators from one-way functions”. In: *Computational Complexity, 2005. Proceedings. Twentieth Annual IEEE Conference on*. IEEE. 2005, pp. 183–197.

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. “Cryptography in NC 0”. In: *SIAM Journal on Computing* (2006).

[AIK08] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. “On pseudorandom generators with linear stretch in NC 0”. In: *Computational Complexity* 17.1 (2008), pp. 38–69.

[App13] Benny Applebaum. “Pseudorandom generators with long stretch and low locality from random local one-way functions”. In: *SIAM Journal on Computing* 42.5 (2013), pp. 2008–2037.

[Gol11] Oded Goldreich. “Candidate one-way functions based on expander graphs”. In: *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*. Springer, 2011, pp. 76–87.

[MST06] Elchanan Mossel, Amir Shpilka, and Luca Trevisan. “On ε-biased generators in NC0”. In: *Random Structures & Algorithms* 29.1 (2006), pp. 56–81.

[LMN93] Nathan Linial, Yishay Mansour, and Noam Nisan. “Constant depth circuits, Fourier transform, and learnability”. In: *Journal of the ACM* (1993), pp. 607–620.

[IW97] R. Impagliazzo and A. Wigderson. “P = BPP if E Requires Exponential Circuits: Derandomizing the XOR Lemma”. In: *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*. 1997, pp. 220–229.

[NW94] Noam Nisan and Avi Wigderson. “Hardness vs Randomness”. In: *Journal of Computer and System Sciences* 49.2 (1994), pp. 149–167.

[Bab+93] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. “BPP Has Subexponential Simulation Unless EXPTIME Has Publishable Proofs”. In: *Computational Complexity* 3 (1993), pp. 307–318.

[Imp95] Russell Impagliazzo. “Hard-core distributions for somewhat hard problems”. In: *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*. IEEE. 1995, pp. 538–545.

[RRV99] R. Raz, O. Reingold, and S. Vadhan. “Error Reduction for Extractors”. In: *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*. 1999, pp. 191–201.

[Zuc97] D. Zuckerman. “Randomness-Optimal Oblivious Sampling”. In: *Random Structures and Algorithms* 11 (1997), pp. 345–367.

[AL93] M. Ajtai and N. Linial. “The Influence of Large Coalitions”. In: *Combinatorica* 13.2 (1993), pp. 129–145.

[CZ16] Eshan Chattopadhyay and David Zuckerman. “Explicit Two-Source Extractors and Resilient Functions”. In: *Proceedings of the 48th Annual ACM Symposium on Theory of Computing*. 2016.

[Mek15] Raghu Meka. “Explicit resilient functions matching Ajtai-Linial”. In: *CoRR* abs/1509.00092 (2015). URL: http://arxiv.org/abs/1509.00092.

[Li16] Xin Li. “Improved two-source extractors, and affine extractors for polylogarithmic entropy”. In: *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science*. 2016.

[Li12] Xin Li. “Design Extractors, Non-Malleable Condensers and Privacy Amplification”. In: *Proceedings of the 44th Annual ACM Symposium on Theory of Computing*. 2012, pp. 837–854.

[GRS04] Ariel Gabizon, Ran Raz, and Ronen Shaltiel. “Deterministic Extractors for Bit-Fixing Sources by Obtaining an Independent Seed”. In: *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*. 2004, pp. 394–403.

[Nis92] Noam Nisan. “Pseudorandom generators for space-bounded computation”. In: *Combinatorica* 12 (1992), pp. 449–461.

[INW94] R. Impagliazzo, N. Nisan, and A. Wigderson. “Pseudorandomness for Network Algorithms”. In: *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*. 1994, pp. 356–364.

[AB09] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[Vad12] Salil P Vadhan. *Pseudorandomness*. Now, 2012.

[Gol+07] Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N Rothblum. “Verifying and decoding in constant depth”. In: *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. ACM. 2007, pp. 440–449.

[O’D14] Ryan O’Donnell. *Analysis of boolean functions*. Cambridge University Press, 2014.

[Vad04] Salil P. Vadhan. “Constructing Locally Computable Extractors and Cryptosystems in the Bounded-Storage Model”. In: *J. Cryptology* 17.1 (2004), pp. 43–77. URL: http://dx.doi.org/10.1007/s00145-003-0237-x.

[Hea08] Alexander D Healy. “Randomness-efficient sampling within NC1”. In: *Computational Complexity* 17.1 (2008), pp. 3–37.

[IZ89] R. Impagliazzo and D. Zuckerman. “How to Recycle Random Bits”. In: *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*. 1989, pp. 248–253.

[RRV02] Ran Raz, Omer Reingold, and Salil Vadhan. “Extracting all the Randomness and Reducing the Error in Trevisan’s Extractors”. In: *JCSS* 65.1 (2002), pp. 97–128.

[Gol95] Oded Goldreich. “Three XOR-Lemmas - An Exposition”. In: *Electronic Colloquium on Computational Complexity (ECCC)* 2.56 (1995).

[BIW06] Boaz Barak, Russell Impagliazzo, and Avi Wigderson. “Extracting randomness using few independent sources”. In: *SIAM Journal on Computing* 36.4 (2006), pp. 1095–1118.

[BQ12] Andrej Bogdanov and Youming Qiao. “On the security of GoldreichâĂŹs one-way function”. In: *computational complexity* 21.1 (2012), pp. 83–127.

[CM01] Mary Cryan and Peter Bro Miltersen. “On pseudorandom generators in NC0”. In: *Mathematical Foundations of Computer Science 2001*. Springer, 2001, pp. 272–284.

[Lin16] Huijia Lin. “Indistinguishability obfuscation from constant-degree graded encoding schemes”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2016, pp. 28–57.

[NR99] Moni Naor and Omer Reingold. “Synthesizers and Their Application to the Parallel Construction of Pseudo-Random Functions”. In: *J. Comput. Syst. Sci.* 58.2 (1999), pp. 336–375. DOI: 10.1006/jcss.1998.1618. URL: https://doi.org/10.1006/jcss.1998.1618.

[MV15] Eric Miles and Emanuele Viola. “Substitution-Permutation Networks, Pseudorandom Functions, and Natural Proofs”. In: *J. ACM* 62.6 (2015), 46:1–46:29. DOI: 10.1145/2792978. URL: http://doi.acm.org/10.1145/2792978.

[BL16] Andrej Bogdanov and Chin Ho Lee. “Homomorphic evaluation requires depth”. In: *Theory of Cryptography Conference*. Springer. 2016, pp. 365–371.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. “Function Secret Sharing.” In: *EUROCRYPT (2)*. 2015, pp. 337–367.

[Bog+16] Andrej Bogdanov, Yuval Ishai, Emanuele Viola, and Christopher Williamson. “Bounded indistinguishability and the complexity of recovering secrets”. In: *Annual Cryptology Conference*. Springer. 2016, pp. 593–618.

[Sha79] Adi Shamir. “How to share a secret”. In: *Communications of the ACM* 22.11 (1979), pp. 612–613.

[BT17] Mark Bun and Justin Thaler. “A Nearly Optimal Lower Bound on the Approximate Degree of AC0”. In: *FOCS*. 2017.

[BW17] Andrej Bogdanov and Christopher Williamson. “Approximate Bounded Indistinguishability.” In: *International Colloquium on Automata, Languages, and Programming*. 2017.

[MP88] Marvin Minsky and Seymour Papert. *Perceptrons*. MIT press, 1988.

[GS16] Venkatesan Guruswami and Adam Smith. “Optimal Rate Code Constructions for Computationally Simple Channels”. In: *Journal of the ACM (JACM)* 63.4 (2016), p. 35.

[LV11] Shachar Lovett and Emanuele Viola. “Bounded-depth circuits cannot sample good codes”. In: *Computational Complexity (CCC), 2011 IEEE 26th Annual Conference on*. IEEE. 2011, pp. 243–251.

[LV15] Chin Ho Lee and Emanuele Viola. “Some limitations of the sum of small-bias distributions.” In: *Electronic Colloquium on Computational Complexity (ECCC)*. Vol. 22. Citeseer. 2015, p. 5.

[Smi07] Adam D Smith. “Scrambling adversarial errors using few random bits, optimal information reconciliation, and better private codes”. In: *Symposium on Discrete Algorithms: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Vol. 7. 09. 2007, pp. 395–404.

[Hem+11] Brett Hemenway, Rafail Ostrovsky, Martin Strauss, and Mary Wootters. “Public key locally decodable codes with short keys”. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques* (2011), pp. 605–615.

[MV91] Yossi Matias and Uzi Vishkin. “Converting high probability into nearly-constant timeâĂŤwith applications to parallel hashing”. In: *Proceedings of the twenty-third annual ACM symposium on Theory of computing*. ACM. 1991, pp. 307–316.

[Hag91] Torben Hagerup. “Fast parallel generation of random permutations”. In: *International Colloquium on Automata, Languages, and Programming*. Springer. 1991, pp. 405–416.

[Vio12] Emanuele Viola. “the complexity of distributions”. In: *SIAM Journal on Computing* 41 (2012), pp. 191–218.

[Che+07] Hao Chen, Ronald Cramer, Shafi Goldwasser, Robbert De Haan, and Vinod Vaikuntanathan. “Secure computation from random error correcting codes”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2007, pp. 291–310.

[Che16] Mahdi Cheraghchi. “Nearly optimal robust secret sharing”. In: *Information Theory (ISIT), 2016 IEEE International Symposium on*. IEEE. 2016, pp. 2509–2513.

[Vio09] Emanuele Viola. “On approximate majority and probabilistic time”. In: *Computational Complexity* 18.3 (2009), p. 337.

[BBR88] Charles H Bennett, Gilles Brassard, and Jean-Marc Robert. “Privacy amplification by public discussion”. In: *SIAM journal on Computing* 17.2 (1988), pp. 210–229.

[Lip94] Richard J. Lipton. "A new approach to information theory". In: *STACS 94: 11th Annual Symposium on Theoretical Aspects of Computer Science Caen, France, February 24–26, 1994 Proceedings*. Ed. by Patrice Enjalbert, Ernst W. Mayr, and Klaus W. Wagner. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 699–708. ISBN: 978-3-540-48332-8. DOI: 10.1007/3-540-57785-8_183. URL: https://doi.org/10.1007/3-540-57785-8_183.

[CL16] Kuan Cheng and Xin Li. "Randomness Extraction in AC0 and with Small Locality". In: *arXiv preprint arXiv:1602.01530* (2016).

[Alo+92a] Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ron M Roth. "Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs". In: *IEEE Transactions on information theory* 38.2 (1992), pp. 509–516.

[Dam+08] Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam Smith. "Scalable multiparty computation with nearly optimal work and resilience". In: *Annual International Cryptology Conference*. Springer. 2008, pp. 241–261.

[AD11] Anne Auger and Benjamin Doerr. *Theory of randomized search heuristics: Foundations and recent developments*. Vol. 1. World Scientific, 2011.

[Bis+16] Allison Bishop, Valerio Pastro, Rajmohan Rajaraman, and Daniel Wichs. "Essentially optimal robust secret sharing with maximal corruptions". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2016, pp. 58–86.

[KNR05] Eyal Kaplan, Moni Naor, and Omer Reingold. "Derandomized constructions of k-wise (almost) independent permutations". In: *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2005, pp. 354–365.

[HVLP98] Tom Høholdt, Jacobus H Van Lint, and Ruud Pellikaan. "Algebraic geometry codes". In: *Handbook of coding theory* 1.Part 1 (1998), pp. 871–961.

[Orl91] A. Orlitsky. "Interactive communication: balanced distributions, correlated files, and average-case complexity". In: *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*. 1991, pp. 228–238. DOI: 10.1109/SFCS.1991.185373.

[Cor+00] Graham Cormode, Mike Paterson, Suleyman Cenk Sahinalp, and Uzi Vishkin. “Communication complexity of document exchange”. In: *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM. 2000, pp. 197–206.

[IMS05] Utku Irmak, Svilen Mihaylov, and Torsten Suel. “Improved single-round protocols for remote file synchronization”. In: *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*. Vol. 3. IEEE. 2005, pp. 1665–1676.

[Jow12] Hossein Jowhari. “Efficient Communication Protocols for Deciding Edit Distance”. In: *ESA*. 2012.

[CGK16] Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. “Low Distortion Embedding from Edit to Hamming Distance using Coupling”. In: *Proceedings of the 48th IEEE Annual Annual ACM SIGACT Symposium on Theory of Computing*. ACM. 2016.

[BQ16] Djamal Belazzougui and Zhang Qin. “Edit Distance: Sketching, Streaming, and Document Exchange”. In: *Proceedings of the 57th IEEE Annual Symposium on Foundations of Computer Science*. IEEE. 2016, pp. 51–60.

[Bel15] Djamal Belazzougui. “Efficient Deterministic Single Round Document Exchange for Edit Distance”. In: *CoRR* abs/1511.09229 (2015).

[Lev66] V. I. Levenshtein. “Binary Codes Capable of Correcting Deletions, Insertions and Reversals”. In: *Soviet Physics Doklady* 10 (1966), p. 707.

[RRV65] G. M. Tenengol’ts R. R. Varshamov. “Code Correcting Single Asymmetric Errors”. In: *Avtomat. i Telemekh* 26 (1965), pp. 288–292.

[SZ99] L. J. Schulman and D. Zuckerman. “Asymptotically good codes correcting insertions, deletions, and transpositions”. In: *IEEE Transactions on Information Theory* 45.7 (1999), pp. 2552–2557. ISSN: 0018-9448. DOI: 10.1109/18.796406.

[MBT10] H. Mercier, V. K. Bhargava, and V. Tarokh. “A survey of error-correcting codes for channels with symbol synchronization errors”. In: *IEEE Communications Surveys Tutorials* 12.1 (2010), pp. 87–96. ISSN: 1553-877X. DOI: 10.1109/SURV.2010.020110.00079.

[GW17] V. Guruswami and C. Wang. “Deletion Codes in the High-Noise and High-Rate Regimes”. In: *IEEE Transactions on Information Theory* 63.4 (2017), pp. 1961–1970. ISSN: 0018-9448. DOI: 10.1109/TIT.2017.2659765.

[GL16] V. Guruswami and R. Li. “Efficiently decodable insertion/deletion codes for high-noise and high-rate regimes”. In: *2016 IEEE International Symposium on Information Theory (ISIT)*. 2016, pp. 620–624. DOI: 10.1109/ISIT.2016.7541373.

[BG16] Boris Bukh and Venkatesan Guruswami. “An improved bound on the fraction of correctable deletions”. In: *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*. ACM. 2016, pp. 1893–1901.

[HS17] Bernhard Haeupler and Amirbehshad Shahrasbi. “Synchronization strings: codes for insertions and deletions approaching the Singleton bound”. In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. ACM. 2017, pp. 33–46.

[HS18] Bernhard Haeupler and Amirbehshad Shahrasbi. “Synchronization Strings: Explicit Constructions, Local Decoding, and Applications”. In: *Proceedings of the 50th Annual ACM Symposium on Theory of Computing*. 2018.

[Che+18] K. Cheng, B. Haeupler, X. Li, A. Shahrasbi, and K. Wu. “Synchronization Strings: Efficient and Fast Deterministic Constructions over Small Alphabets”. In: *ArXiv e-prints* (2018). arXiv: 1803.03530 [cs.IT].

[BGZ17] J. Brakensiek, V. Guruswami, and S. Zbarsky. “Efficient Low-Redundancy Codes for Correcting Multiple Deletions”. In: *IEEE Transactions on Information Theory* PP.99 (2017), pp. 1–1. ISSN: 0018-9448. DOI: 10.1109/TIT.2017.2746566.

[SS02] D Shapira and J. A. Storer. “Edit distance with move operations”. In: *Proceedings of the 13th Symposium on Combinatorial Pattern Matching*. 2002, pp. 85–98.

[CM07] Graham Cormode and S. Muthukrishnan. “The string edit distance matching problem with moves”. In: *ACM Transactions on Algorithms* 3.1 (2007).

[HSV17] Bernhard Haeupler, Amirbehshad Shahrasbi, and Ellen Vitercik. “Synchronization Strings: Channel Simulations and Interactive Coding for Insertions and Deletions”. In: *arXiv preprint arXiv:1707.04233* (2017).

[Alo+92b] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. “Simple Constructions of Almost k-wise Independent Random Variables”. In: *Random Structures & Algorithms* 3.3 (1992), pp. 289–304.

[De+10] Anindya De, Omid Etesami, Luca Trevisan, and Madhur Tulsiani. “Improved pseudorandom generators for depth 2 circuits”. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2010, pp. 504–517.

[Alo+95] Noga Alon, Uriel Feige, Avi Wigderson, and David Zuckerman. “Derandomized graph products”. In: *Computational Complexity* 5.1 (1995), pp. 60–75.

[HLW06] Shlomo Hoory, Nathan Linial, and Avi Wigderson. “Expander graphs and their applications”. In: *Bulletin of the American Mathematical Society* 43.4 (2006), pp. 439–561.

Kuan Cheng

Johns Hopkins University, Computer Science Department.

3400 N. Charles St.
Baltimore, MD 21218
+1 (443) 676-8032
kcheng17@jhu.edu
webpage

EDUCATION

Johns Hopkins University — *PhD*

Sept. 2014 - NOW. Computer Science Department, advised by Professor Xin Li.

Research Field: Randomness in Computation, Circuit Complexity, Coding Theory.

Tsinghua University — *Master of Engineering*

Sept. 2011 - June 2014. Computer Science Department.

Specified Field: Algorithms, Cryptography.

Shandong University — *Bachelor of Engineering*

Sept. 2007 - June 2011. Software Engineering Department.

SELECTED PUBLICATIONS

Kuan Cheng, Zhengzhong Jin, Xin Li, and Ke Wu. "Block Edit Errors with Transpositions: Deterministic Document Exchange Protocols and Almost Optimal Binary Codes. " In International Colloquium on Automata, Languages and Programming (ICALP) 2019.

Kuan Cheng, Bernhard Haeupler, Xin Li, Amirbehshad Shahrasbi, Ke Wu. "Synchronization Strings: Efficient and Fast Deterministic Constructions over Small Alphabets." In ACM-SIAM Symposium on Discrete Algorithms (SODA) 2019.

Kuan Cheng, Zhengzhong Jin, Xin Li, and Ke Wu. "Deterministic Document Exchange Protocols, and Almost Optimal Binary Codes for Edit Errors." In Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS). IEEE, 2018.

Kuan Cheng and Xin Li. "Randomness Extraction in AC0 and with Small Locality." Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (Random), 2018.

Kuan Cheng, Yuval Ishai, and Xin Li. "Near-Optimal Secret Sharing and Error Correcting Codes in AC0." In *Theory of Cryptography Conference (TCC)*, pp. 424-458. Springer, Cham, 2017.

COMPETITIONS

2011, The 35th ACM-International Collegiate Programming Contest, Tianjin Site — Bronze Medal;

2010, Mathematical Contest in Modeling (MCM)—Meritorious Winner;

2009, China Undergraduate Mathematical Contest in Modeling (CUMCM) — National First Prize;

AWARDS

2014, JHU CS Dept. Graduate Student Fellowship.

2012, Tsinghua University ESS scholarship.

2010, President Scholarship of Shandong University, Software College.

2009, National Scholarship.

2009, President Scholarship of Shandong University.

2008-2009, First Class Scholarship of Shandong University.

Teaching Experience

Theory of Computation, TA, 2015 Fall, 2016 Fall, 2019 Spring.
Automata & Computation Theory, TA, 2016 Spring, 2017 spring, 2018 Fall.