Cryptography in Constant Parallel Time

Benny Applebaum

Cryptography in Constant Parallel Time

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Benny Applebaum

Submitted to the Senate of
the Technion — Israel Institute of Technology
Tamuz 5767 Haifa June 2007

The research thesis was done under the supervision of Assoc. Prof. Yuval Ishai and Prof. Eyal Kushilevitz in the Computer Science Department.

Acknowledgements

I would like to sincerely thank my supervisors Yuval Ishai and Eyal Kushilevitz. During the last years, I have spent many hours in conversations with Yuval and Eyal. These long discussions have taught me invaluable lessons regarding many aspects of the scientific work and it is hard to overestimate their effect on my scientific perspective. (Of course, any errors or misstatements in the end result are my responsibility . . .) I consider myself lucky to have been guided by Yuval and Eyal, and look forward to collaborate with them in the future.

I was also fortunate to have Oded Goldreich as the editor of the journal versions of some of the works that appear in this dissertation. I am thankful to Oded for his numerous suggestions and comments that have significantly improved this manuscript in many ways.

During my graduate studies, I had the opportunity to discuss research topics with many different people. These interactions have been extremely enjoyable and helped me to increase my knowledge and, more typically, decrease my misconceptions. For this, I would like to thank Omer Barkol, Rotem Bennet, Eli Ben-Sasson, Eli Biham, Iftach Haitner, Danny Harnik, Moni Naor, Erez Petrank, Omer Reingold, Ronny Roth, Amir Shpilka, Amnon Ta-Shma, Enav Weinreb and Emanuele Viola. I am also thankful to Eli Biham, Oded Goldreich, Erez Petrank, and Omer Reingold for serving on my thesis committee.

I would like to thank all the people from the Computer Science department in the Technion, with whom I worked and studied, for making my time at the Technion so pleasant. Special thanks go to my office partner, Boris Kapchits, and my floor mates, Rotem Bennet, Niv Buchbinder, Oren Katzengold, Jonathan Naor and Sharon Shoham. I really liked all these endless coffee breaks!

Finally, I am grateful to my family and friends for making my life joyful and meaningful. Most significantly, I would like to thank my parents, Arie and Elka, for their unlimited love and support; to my brother, Nadav, and my sister, Edna, for showing me some of the roads I have not taken; to my beloved wife, Hilla, for her sensitivity and wisdom and for sharing her life with me; and to my daughter, Shira, for making me feel the happiest person on earth.

The generous financial help of the Fein and Gutwirth foundations is gratefully acknowledged.

Contents

Abstract **1**

1 Introduction **3**
1.1 The Basic Question . . . 3
1.1.1 Further Perspectives . . . 4
1.2 Our Research . . . 5
1.2.1 Results and Organization . . . 6

2 Preliminaries and Definitions **10**
2.1 General . . . 10
2.2 Statistical and Computational indistinguishability . . . 11
2.2.1 Some useful facts . . . 11
2.3 Computational Models . . . 12
2.4 Output locality, input locality and degree . . . 14

3 Randomized Encoding of Functions **17**
3.1 Definitions . . . 17
3.2 Basic Properties . . . 21
3.3 More Aspects of Randomized Encoding . . . 23
3.3.1 The Necessity of Randomness . . . 23
3.3.2 The Power of Randomized Encoding . . . 24
3.3.3 Lower Bounds for Locality and Degree . . . 26

4 Cryptography in NC^0 **28**
4.1 Introduction . . . 28
4.1.1 Previous Work . . . 28
4.1.2 Our Results . . . 29
4.1.3 Overview of Techniques . . . 30
4.1.4 Organization . . . 33
4.2 Randomized Encoding in NC^0 for Functions in $\oplus\mathrm{L}/poly$,$\mathrm{NL}/poly$ . . . 33
4.3 A Generalization of the Locality Construction . . . 37
4.4 One-Way Functions in NC^0 . . . 38
4.4.1 Key Lemmas . . . 39
4.4.2 Main Results . . . 42
4.5 Pseudorandom Generators in NC^0 . . . 43

4.5.1 Cryptographic Generators 44
4.5.2 ε-Biased Generators 47
4.5.3 Generators for Space-Bounded Computation 49
4.5.4 Pseudorandom Generators — Conclusion 50
4.6 Collision-Resistant Hashing in NC^0 50
4.7 Encryption in NC^0 52
4.7.1 Main Results 52
4.7.2 On Decryption in NC^0 54
4.7.3 Security against CPA, CCA1 and CCA2 Attacks 55
4.8 Other Cryptographic Primitives 56
4.8.1 Signatures 56
4.8.2 Commitments 57
4.8.3 Zero-Knowledge Proofs 60
4.8.4 Instance hiding schemes 62
4.9 Summary and Discussion 62
4.9.1 The case of PRFs 62
4.9.2 Open Problems 63

5 Computationally Private Randomizing Polynomials and Their Applications 65
5.1 Introduction 65
5.1.1 Overview of Results and Techniques 65
5.1.2 Organization 68
5.2 Computational Encoding in NC^0 for Efficiently Computable Functions 69
5.2.1 From PRG to One-Time Encryption 69
5.2.2 From One-Time Encryption to Computational Encoding 71
5.2.3 Main Results 74
5.2.4 Proof of Lemma 5.2.11 76
5.3 Applications 80
5.3.1 Relaxed Assumptions for Cryptography in NC^0 80
5.3.2 Parallel Reductions between Cryptographic Primitives 82
5.3.3 Secure Multi-Party Computation 84

6 On Pseudorandom Generators with Linear Stretch in NC^0 89
6.1 Introduction 89
6.1.1 Our Contribution 90
6.1.2 Related Work 91
6.2 Preliminaries 92
6.2.1 Some useful facts 92
6.3 LPRG in NC^0 implies Hardness of Approximation 92
6.4 Using NC^0 Implementations of Other Cryptographic Primitives 94
6.4.1 Pseudoentropy Generator 95
6.4.2 String Commitment 96
6.5 A Construction of LPRG in NC^0 97
6.5.1 Overview 97
6.5.2 The Assumption 98
6.5.3 The Construction 100

6.5.4 ε-Biased Generators in Uniform NC^0 103
6.5.5 Alekhnovich's Assumption Implies Assumption 6.5.1 105
6.6 The Necessity of Expansion 107
6.6.1 Actual Results 107
6.6.2 Discussion 108

7 Cryptography with Constant Input Locality 110
7.1 Introduction 110
7.1.1 Our Results 111
7.1.2 Our Techniques 112
7.1.3 Previous Work 112
7.2 Preliminaries 113
7.3 Randomized Encoding with Constant Input Locality 114
7.4 Primitives with Constant Input Locality and Output Locality 117
7.4.1 Main Assumption: Intractability of Decoding Random Linear Code 117
7.4.2 Pseudorandom Generator in Local_3^3 119
7.4.3 Commitment in Local_3^4 122
7.4.4 Semantically Secure Public-Key Encryption in $\mathrm{Local}_3^{O(1)}$ 123
7.5 Negative Results for Cryptographic Primitives 125
7.5.1 Basic Observations 125
7.5.2 MACs and Signatures 126
7.5.3 Non-Malleable Encryption 127
7.5.4 The Impossibility of Implementing a PRG in Local_2 128
7.6 Negative Results for Randomized Encodings 129
7.6.1 A Necessary Condition for Encoding with Low Input Locality 129
7.6.2 Impossibility of Universal Encoding for Linear Functions 131
7.7 Conclusions and Open Questions 132

8 One-Way Functions with Optimal Output Locality 133
8.1 Introduction 133
8.1.1 Semi-Private Randomized Encoding and Robust OWF 134
8.1.2 Constructing Robust OWF 134
8.2 Preliminaries 135
8.2.1 Semi-Private Randomized Encoding 135
8.2.2 One-Way Functions 135
8.3 From Robust OWF to OWF in NC_3^0 137
8.3.1 A Warmup 137
8.3.2 The Actual Construction 138
8.4 A Candidate Robust One-Way Function 140

A On Collections of Cryptographic Primitives 144

List of Figures

1.2.1 New NC^0 reductions 9

2.3.1 An example of a branching program 13
2.3.2 Complexity classes inside P 14

4.2.1 The matrices $R_1(r^{(1)}), L(x)$ and $R_2(r^{(2)})$ 34

6.6.1 A trivial “slightly” unbalanced expander 109

Abstract

We study the parallel time-complexity of basic cryptographic primitives. Specifically, we consider the possibility of computing instances of these primitives by NC^0 circuits, in which each output bit depends on a constant number of input bits. Despite previous efforts in this direction, there has been no convincing theoretical evidence supporting this possibility, which was posed as an open question in several previous works (e.g., [Hås87, Gol00, CM01, KL01, MST03]). We essentially settle this question by providing strong evidence for the possibility of cryptography in NC^0. In particular, we derive the following results:

Cryptographic primitives in NC^0. We show that many cryptographic primitives can be realized in NC^0 under standard intractability assumptions used in cryptography, such as ones related to factoring, discrete logarithm, or lattice problems. This includes one-way functions, pseudorandom generators, symmetric and public key encryption schemes, digital signatures, message authentication schemes (MACs), commitment schemes, collision resistant hash functions and zero-knowledge proofs. Moreover, we provide a *compiler* that transforms an implementation of a cryptographic primitive in a relatively "high" complexity class into an NC^0 implementation. This compiler is also used to derive new (unconditional) NC^0 *reductions* between different cryptographic primitives. In some cases, no parallel reductions of this type were previously known, even in NC. Interestingly, we get *non-black-box* reductions.

Pseudorandom generators with linear stretch in NC^0. We construct an NC^0 pseudorandom generator that stretches n random bits into cn pseudorandom bits, for some constant $c > 1$. The security of this pseudorandom generator relies on a relatively new intractability assumption presented by Alekhnovich [Ale03]. (PRGs obtained by the general compiler described above were limited to stretching a seed of n bits to $n + o(n)$ pseudorandom bits.) We also identify a new connection between such pseudorandom generators and hardness of approximations for combinatorial optimization problems. In particular, we show that an NC^0 pseudorandom generator with large (linear) stretch implies that Max 3SAT cannot be efficiently approximated to within some multiplicative constant. Our argument is quite simple and does not rely on PCP machinery.

Cryptography with constant input locality. We study the possibility of carrying out cryptographic tasks by functions in which each input bit affects a constant number of *output* bits, i.e., functions with constant *input* locality. (Our previous results have only addressed the case of a constant *output* locality, which does not imply a constant *input* locality.) We (almost) characterize what cryptographic tasks can be performed with constant input locality. On the negative side, we show that primitives which require some form of non-malleability (such as digital signatures, message authentication, or non-malleable encryption) *cannot* be realized with constant input locality. On the positive side, assuming the intractability of certain problems from the domain of error correcting codes (namely, hardness of decoding a random linear code or the security of the McEliece cryptosystem), we obtain new constructions of one-way functions, pseudorandom generators, commitments, and semantically-secure public-key encryption schemes whose input locality is constant. Moreover, these constructions also enjoy constant *output locality.* Therefore, they give rise to cryptographic hardware that has constant-depth, constant fan-in and constant *fan-out.*

Chapter 1

Introduction

1.1 The Basic Question

Cryptography is concerned with communication and computation in the presence of adversaries. In the last few decades, the theory of cryptography has been extensively developed and has successfully provided solutions to many cryptographic challenges. Moreover, due to the evolvement of the Internet, cryptographic tools are now widely employed both by individuals and by organizations. Daily actions, such as checking an account balance or electronic commerce, make an essential use of cryptographic primitives such as encryption schemes and digital signatures.

In this research, we consider the question of minimizing the computational complexity of basic cryptographic primitives such as one-way functions (functions that are easy to compute but are hard to invert), pseudorandom generators (functions that stretch a short random seed into a longer "random-looking" string) [BM84, Yao82], secure encryption schemes, digital signatures, and others. Pushing this question to an extreme, it is natural to ask whether such primitives can be made "computationally simple" to the extent that each bit of their output is only influenced by a constant number of input bits, independently of the desired level of security. Specifically, the following fundamental question was posed in several previous works (e.g., [Hås87, Gol00, CM01, KL01, MST03]):

> Is it possible to compute one-way functions, or even pseudorandom generators so that every bit of the output can be computed by reading a constant number of bits of the input?

The class of functions which are computationally simple in the above sense is denoted by NC^0. We let NC^0_c denote the class of NC^0 functions in which each of the output bits depends on at most c input bits, and refer to the constant c as the *output locality* of the function (or locality for short).

The above question is qualitatively interesting as it explores the possibility of obtaining cryptographic hardness using "extremely simple" functions. However, functions in NC^0 might be considered to be too degenerate to perform any interesting computational tasks, let alone cryptographic tasks which are perceived to be inherently complex. Indeed, all common implementations of cryptographic primitives not only require each output bit to depend on many inputs bits, but also involve rather complex manipulations of these bits.

The possibility of cryptography in NC^0 has been studied since the mid-eighties. Several works have made progress in related directions [Hås87, GKY89, LMN93, YY94, IN96, NR99, NR04, Gol00, CM01, MST03, Vio05], conjecturing either the existence (e.g., [Gol00]) or the non-existence (e.g., [CM01]) of cryptographic primitives in NC^0. However, despite all this body of work there has been no significant theoretical evidence supporting either a positive or a negative conclusion.

In this dissertation, we provide a surprising affirmative answer to this question. We prove that most cryptographic primitives can be implemented by NC^0 functions under standard intractability assumptions commonly used in cryptography (e.g., that factoring large integers is computationally hard). Specifically, primitives such as one-way functions, encryption, digital signatures, and others can be computed by extremely "simple" functions, in which every bit of the output depends on only four bits of the input.

This result is of both theoretical and practical interest. From a theoretical point of view, it is part of a general endeavor to identify the minimal resources required for carrying out natural computational tasks. From a more practical point of view, an NC^0 implementation of a cryptographic primitive supports an ultimate level of parallelism: in an NC^0 function, different output bits can be computed in parallel without requiring intermediate sequential computations. In particular, such functions can be computed in *constant* parallel time, i.e., by constant-depth circuits with bounded fan-in. Thus, NC^0 primitives may give rise to super-fast cryptographic hardware.

1.1.1 Further Perspectives

It is instructive to examine the question of cryptography in NC^0 from two distinct perspectives:

- (Applied cryptography) In the community of applied cryptography it is widely accepted that functions with low locality should not be used as block ciphers or hash functions. Indeed, several central design principles of block ciphers (e.g., so-called Confusion-Diffusion [Sha49], Avalanche Criterion [Fei73], Completeness [KD79] and Strict Avalanche Criterion [WT86]) explicitly state that the input-output dependencies of a block cipher should be complex. In particular, in his seminal paper Feistel asserts that: "The important fact is that all output digits have potentially become very involved functions of all input digits" [Fei73]. (In fact, this concern dates back to Shannon [Sha49].) It is easy to justify this principle in the context of block-ciphers (which are theoretically modeled as pseudorandom functions or permutations), but it is not clear whether it is also necessary in other cryptographic applications (e.g., one-way functions, pseudorandom generators, or probabilistic public-key encryption schemes).

- (Complexity theory) The possibility of cryptography in NC^0 is closely related to the intractability of Constraint Satisfaction Problems. Inverting a function in NC^0_c can be formulated as a Constraint Satisfaction Problem in which each constraint involves at most c variables (c-CSP). For example, finding an inverse of a string $y \in \{0,1\}^n$ under a function $f : \{0,1\}^n \to \{0,1\}^n$ is equivalent to solving a CSP problem over n variables $x = x_1, \ldots, x_n$ of the form:

$$\begin{cases} y_1 = f_1(x), \\ \quad \vdots \\ y_n = f_n(x), \end{cases}$$

  where f_i is the function that computes the i-th output bit of f and y_i is the i-th bit of y. If f is in, say, NC^0_4 we get an instance of a 4-CSP problem. Constraint satisfaction problems are well studied in complexity theory and are known to be "hard" in several aspects. In particular, the Cook-Levin theorem [Coo71, Lev73] shows that it is NP-hard to exactly solve

3-CSP problems, while the PCP theorem [ALM$^+$98, AS98] shows that it is NP-hard even to find an approximate solution. It should be noted that, for several reasons, NP-hardness does not imply cryptographic hardness. Hence, although these results might indicate that it is not easy to invert NC^0 functions in the worst case, they fall short of proving the existence of one-way functions in NC^0.

1.2 Our Research

Our main goal is to draw the theoretical and practical limitations of cryptography in constant parallel-time. Hence, we try to characterize the precise computational power needed to fulfil different cryptographic tasks. In particular, we will be interested in questions of the form: Can a cryptographic primitive $\mathcal{P}$ be realized in some low complexity class WEAK? If so, what are the minimal assumptions required for such an implementation? We usually instantiate these meta-questions with the class NC^0, but we will also consider other complexity classes, such as sub-classes of NC^0 in which the output locality is bounded by some specific constant (e.g., NC^0_3), or the class of functions whose *input locality* is constant (i.e., functions in which each bit of the input affects on a constant number of output bits).

Our approach

Our key observation is that instead of computing a given "cryptographic" function f, it might suffice to compute a related function $\hat{f}$ which (1) preserves the cryptographic properties of f; and (2) admits an efficient implementation. To this end, we rely on the machinery of *randomized encoding*, which was introduced in [IK00] (under the algebraic framework of *randomizing polynomials*). A randomized encoding of a function $f(x)$ is a randomized mapping $\hat{f}(x,r)$ whose output distribution depends only on the output of f. Specifically, it is required that: (1) there exists a decoder algorithm that recovers $f(x)$ from $\hat{f}(x,r)$, and (2) there exists a simulator algorithm that given $f(x)$ samples from the distribution $\hat{f}(x,r)$ induced by a uniform choice of r. That is, the distribution $\hat{f}(x,r)$ hides all information about x except for the value $f(x)$.

We show that the security of most cryptographic primitives is inherited by their randomized encoding. This gives rise to the following paradigm. Suppose that we want to construct some cryptographic primitive $\mathcal{P}$ in some low complexity class WEAK. Then, we can try to encode functions from a higher complexity class STRONG by functions from WEAK. Now, if we have an implementation f of the primitive $\mathcal{P}$ in STRONG, we can replace f by its encoding $\hat{f} \in \mathsf{WEAK}$ and obtain a low-complexity implementation of $\mathcal{P}$. This approach is extensively used in this dissertation.

Non-black-box techniques. In order to encode a function f, we apply a "compiler" to the *description* of f (given in some computational model). This technique is inherently *non-black-box* and, in some cases, it also yields parallel non-black-box transformations between different primitives. That is, the "code" of the NC^0-reduction we get, implementing a primitive $\mathcal{P}$ using an oracle to a primitive $\mathcal{P}'$, depends on the code of the underlying primitive $\mathcal{P}'$. This should be contrasted with most known transformations in cryptography, which make a black-box use of the underlying primitive. We believe that our work provides further evidence for the usefulness of non-black-box techniques in cryptography.

1.2.1 Results and Organization

In the following we give an outline of our results. Some of these results are also summarized graphically in Tables 1.2.1, 1.2.1, 1.2.3 and Figure 1.2.1. The material presented in this thesis was obtained in joint works with Yuval Ishai and Eyal Kushilevitz [AIK06b, AIK06a, AIK06c, AIK07, AIK05].

Chapter 2 — Preliminaries. We set the basic notation and definitions used in this dissertation. This includes the notions of statistical and computational indistinguishability and some of their properties, as well as definitions and conventions regarding several computational models. We also define the main complexity classes mentioned in this thesis and investigate some of their simple properties.

Chapter 3 — Randomized Encoding of Functions. We define the main variants of randomized encoding, including several *information-theoretic* variants as well as a *computational* variant. We investigate several useful properties of this notion, and discuss some of its limitations. Most of the material in this chapter is based on [AIK06b, Section 4].

Chapter 4 — Cryptography in NC^0. We show that randomized encoding preserves the security of many cryptographic primitives. We also construct an (information-theoretic) encoding in NC^0_4 for any function in NC^1 or even in $\oplus\mathrm{L}/poly$. This result is obtained by relying on the constructions of [IK02] which give a low degree encoding for these classes. The combination of these results gives a compiler that takes as an input a code of an NC^1 implementation of some cryptographic primitive and generates an NC^0_4 implementation of the same primitive. This works for many cryptographic primitives such as OWFs, PRGs, one-way permutations, trapdoor-permutations, collision-resistant hash functions, encryption schemes, message authentication schemes, digital signatures, commitments and zero-knowledge proofs. The existence of many of the above primitives in NC^1 is a relatively mild assumption, implied by most number-theoretic or algebraic intractability assumptions commonly used in cryptography. We remark that in the case of two-party primitives (e.g., encryption schemes, signatures, commitments, zero-knowledge proofs) our transformation results in an NC^0 sender (i.e., the encrypting party, committing party, signer or prover) but does not promise anything regarding the parallel complexity of the receiver (the decrypting party or verifier).[1] In fact, we prove that, in all these cases, the receiver *cannot* be implemented by an NC^0 function, regardless of the complexity of the sender. (See Table 1.2.1.) Our techniques can also be applied to obtain unconditional constructions of non-cryptographic PRGs. In particular, building on [MST03], we obtain an ϵ-biased generator in NC^0_3, answering an open question posed in [MST03]. The material in this chapter is mainly based on [AIK06b].

Chapter 5 — Computationally Private Randomized Encoding and its Applications. We consider a relaxed notion of randomized encodings, where the "hiding" property of randomized encoding is relaxed to the computational setting. We construct such an encoding in NC^0_4 for

[1] An interesting feature of the case of commitment is that we can also improve the parallel complexity at the receiver's end. Specifically, it can be implemented by an AC^0 circuit (or even by a weaker circuit family). This feature of commitment carries on to some applications of commitments such as distributed coin-flipping and ZK proofs.

every *polynomial-time* computable function, assuming the existence of a PRG in $\oplus\mathrm{L}/poly$. We present several applications of computationally private randomized encoding. In particular, we considerably relax the sufficient assumptions for NC^0 constructions of cryptographic primitives (see Table 1.2.1), obtain new *unconditional* NC^0 transformations between primitives (see Figure 1.2.1), and simplify the design of constant-round protocols for multiparty computation. This chapter is based on [AIK06a].

Chapter 6 — Pseudorandom Generators with Linear Stretch in NC^0. The aforementioned constructions of PRGs in NC^0 were limited to stretching a seed of n bits to $n + o(n)$ bits. This leaves open the existence of a PRG with a linear (let alone superlinear) stretch in NC^0. We construct a linear-stretch PRG in NC^0_4 under a relatively new intractability assumption presented by Alekhnovich [Ale03]. The linear stretch of this PRG is essentially optimal as there is no PRG with superlinear stretch in NC^0_4 [MST03]. We also show that the existence of a linear-stretch PRG in NC^0 implies non-trivial hardness of approximation results *without relying on PCP machinery.* In particular, it implies (via a simple proof) that Max3SAT is hard to approximate to within some multiplicative constant. The material of this chapter is based on [AIK06c].

Chapter 7 — Cryptography with Constant Input Locality. We study the possibility of carrying out cryptographic tasks by functions in which each input bit affects a constant number of output bits, i.e., functions with constant *input* locality. (Our previous results have only addressed the case of a constant *output* locality, which does not imply a constant *input* locality.) We (almost) characterize what cryptographic tasks can be performed with constant input locality. On the negative side, we show that primitives which require some form of non-malleability (such as digital signatures, message authentication, or non-malleable encryption) *cannot* be realized with constant (or, in some cases, even logarithmic) input locality. On the positive side, assuming the intractability of some problems from the domain of error correcting codes (namely, hardness of decoding a random linear code or the security of the McEliece cryptosystem), we obtain new constructions of OWFs, PRGs, commitments, and semantically-secure public-key encryption schemes whose input locality is constant. (See Table 1.2.1.) Moreover, these constructions also enjoy constant *output locality.* Therefore, they give rise to cryptographic hardware that has constant-depth, constant fan-in and constant *fan-out.* As a byproduct, we also construct a pseudorandom generator whose output and input locality are both optimal (namely, 3). Our positive results rely on a new construction of randomized encoding with constant input locality, while the negative results shed some light on the limitation of such an encoding. This chapter is based on [AIK07].

Chapter 8 — One-way Functions with Optimal Output Locality. In Chapter 4 it is shown that, under relatively mild assumptions, there exist one-way functions (OWFs) in NC^0_4. This result is not far from optimal as there is no OWF in NC^0_2. The gap is partially closed in Chapter 7 by showing that the existence of a OWF (and even a PRG) in NC^0_3 is implied by the intractability of decoding a random linear code. In this chapter we provide further evidence for the existence of OWF in NC^0_3. We construct such a OWF based on the existence of a OWF that enjoys a certain strong “robustness” property. We also show how to construct such a function assuming that a random function of locality $O(\log n)$ is one-way. (A similar assumption was previously made by Goldreich [Gol00].) This result is obtained by constructing a new variant of randomized encoding. This chapter is based on [AIK05].

Primitive	General assumption	Concrete assumption
One-way function	$\exists$ in NL/*poly*, $\oplus$L/*poly*	factoring, DLOG, lattices
One-way permutation	$\exists$ in $\oplus$L/*poly*	LRSA, DLOG
Trapdoor permutation	$\exists$ in $\oplus$L/*poly*	LRSA
Pseudorandom generator	$\exists$ in $\oplus$L/*poly*	factoring, DLOG, lattices
Collision-resistant hashing	$\exists$ in $\oplus$L/*poly*	factoring, DLOG, lattices
Public-key encryption		
Encrypting	$\exists$ in $\oplus$L/*poly*, NL/*poly* OR $\exists$ + EPRG	factoring, DDH, lattices
Decrypting	$\times$	
Symmetric encryption		
Encrypting	$\exists$ in $\oplus$L/*poly*, NL/*poly* OR EPRG	factoring, DLOG, lattices
Decrypting	$\times$	
Signatures, MACs		
Signing	$\exists$ in $\oplus$L/*poly*, NL/*poly* OR EPRG	factoring, DLOG, lattices
Verifying	$\times$	
Non-interactive commitment	$\exists$ in $\oplus$L/*poly*, NL/*poly* OR $\exists$ + EPRG	factoring, DLOG, lattices
2-round stat. hiding commitment		
Committing	$\exists$ in $\oplus$L/*poly*, NL/*poly*	factoring, DLOG, lattices
Verifying	$\times$	
NIZK for NP		
Proving	$\exists$ in $\oplus$L/*poly*, NL/*poly* OR $\exists$ + EPRG	factoring
Verifying	$\times$	
Constant-round ZK proof for NP		
Proving	$\exists$ in $\oplus$L/*poly*, NL/*poly* OR $\exists$ + EPRG	factoring, DLOG, lattices
Verifying	$\times$	

Table 1.2.1: Sufficient conditions for NC^0 implementations of different primitives. In the case of PRGs (resp. collision-resistant hashing) we get an NC^0 implementation with *sublinear* stretch (resp. shrinkage). **General assumptions**: We write "$\exists$ in $\mathcal{C}$" to denote the assumption that the primitive can be realized in the complexity class $\mathcal{C}$. When $\mathcal{C}$ is omitted, we refer to the class P, that is we assume that the primitive can be realized at all. We write "EPRG" to denote the existence of a PRG in $\oplus$L/*poly*. (The class $\oplus$L/*poly* contains the classes L/*poly* and NC^1 and is contained in NC^2. See Section 2.3.) The symbol "$\times$" denote an impossibility result. **Concrete assumptions**: We use DLOG, LRSA, and DDH to denote the intractability of the discrete logarithm problem, the RSA problem with low exponent, and the decisional Diffie-Hellman problem, respectively. For example, the public-key encryption entry states that we can get a scheme in which the encrypting party is in NC^0 under any of the following assumptions: (1) there exists such a scheme in which the encrypting is realized in $\oplus\mathrm{L}/poly \bigcup \mathrm{NL}/poly$; or (2) there exists a PRG in $\oplus$L/*poly* and there exists a public-key encryption scheme at all. Moreover, these assumptions are implied by the intractability of either the factoring problem, the DDH problem or lattice problems. This entry also says that there is no such scheme in which the decryption is realized by an NC^0 function.

Primitive	Assumption
One-way function	code
Pseudorandom generator	code
Non-interactive commitment	code
Public-key encryption	McEliece
Symmetric encryption	code
Signatures, MACs	$\times$
Non-malleable encryption	$\times$

Table 1.2.2: Sufficient conditions for implementations of cryptographic primitives with constant input locality. We use "code" and "McEliece" to denote the intractability of decoding random linear code, and the intractability of inverting the McEliece cryptosystem [McE78], respectively. The symbol $\times$ denotes an impossibility result. The positive results allow an implementation that enjoys constant input locality and constant output locality at the same time.

Stretch	Output Locality	Algebraic Degree	Input Locality	Reference
sublinear	4	3	—	Theorem 4.5.6
linear ✓	4	3	—	Theorem 6.5.11
sublinear	3 ✓	2 ✓	3 ✓	Theorem 7.4.9

Table 1.2.3: Pseudorandom generators: stretch vs. locality. A PRG has sublinear (resp. linear) stretch if it stretches n bits to $n + o(n)$ bits (resp. $n + \Omega(n)$ bits). A parameter is marked as optimal (✓) if when fixing the other parameters it cannot be improved. The first construction requires the existence of a PRG in $\oplus\mathrm{L}/poly$, while the other two are based on concrete assumptions.

$$\begin{tikzcd}
 & & \text{Symmetric-Encryption} \\
\text{1-1 OWF} \arrow[r] \arrow[d] & \text{min-PRG} \arrow[r, Rightarrow] \arrow[ur, Rightarrow] \arrow[dr, Rightarrow] & \text{Commitment} \\
\text{Non-interactive Commitment} & & \text{Signature/MAC}
\end{tikzcd}$$

Figure 1.2.1: New NC^0 reductions. Doubleline arrows denote non-black-box reductions. We write "min-PRG" to denote a pseudorandom generator G with minimal stretch, i.e., $G : \{0,1\}^n \to \{0,1\}^{n+1}$.

Chapter 2

Preliminaries and Definitions

Summary: This chapter presents the basic notation and definitions used in this dissertation. This includes the notions of statistical and computational indistinguishability and some of their properties (Section 2.2), as well as definitions and conventions regarding several computational models (Section 2.3). We also define the main complexity measures mentioned in this thesis, namely, output locality, input locality, and algebraic degree and investigate some of their simple properties (Section 2.4).

2.1 General

Basic notation. Let $\mathbb{N}$ denote the set of positive integers. For a positive integer $n \in \mathbb{N}$, denote by $[n]$ the set $\{1, \ldots, n\}$. For a string $x \in \{0,1\}^*$, let $|x|$ denote the length of x. For a string $x \in \{0,1\}^n$ and an integer $i \in [n]$, let x_i denote the i-th bit of x. Similarly, for $S \subseteq [n]$, let x_S denote the restriction of x to the indices in S. We will write $x_{\oplus i}$ to denote the string x with the i-th bit flipped. For a prime p, let $\mathbb{F}_p$ denote the finite field of p elements. Let $\mathbb{F}$ denote an arbitrary finite field. We will sometimes abuse notation and identify binary strings with vectors over $\mathbb{F}_2$. All vectors will be regarded by default as column vectors. Let $\langle \cdot, \cdot \rangle$ denote inner product over $\mathbb{F}_2$, i.e., for $x, y \in \mathbb{F}_2^n$, $\langle x, y \rangle = \sum_{i=1}^n x_i \cdot y_i$ where arithmetic is over $\mathbb{F}_2$. For a function $f : X \to Y$ and an element $y \in Y$, let $f^{-1}(y)$ denote the set $\{x \in X | f(x) = y\}$. Let $\mathrm{Im}(f)$ denote the set $\left\{y \in Y | f^{-1}(y) \neq \emptyset\right\}$. A function $\varepsilon(\cdot)$ from positive integers to reals is said to be *negligible* if $\varepsilon(n) < n^{-c}$ for any $c > 0$ and sufficiently large n. We will sometimes use $\mathrm{neg}(\cdot)$ to denote an unspecified negligible function.

Probabilistic notation. Let U_n denote a random variable that is uniformly distributed over $\{0,1\}^n$. Different occurrences of U_n in the same statement refer to the same random variable (rather than independent ones). If X is a probability distribution, we write $x \leftarrow X$ to indicate that x is a sample taken from X. Let $\mathrm{support}(X)$ denote the *support* of X (i.e., the set of all elements with non-zero probability), and let $\mathbb{E}(X)$ denote the expectation of X. The *min-entropy* of a random variable X is defined as $\mathrm{H}_\infty(X) \stackrel{\text{def}}{=} \min_x \log(\frac{1}{\Pr[X=x]})$. Let $\mathrm{H}_2(\cdot)$ denote the binary entropy function, i.e., for $0 < p < 1$, $\mathrm{H}_2(p) \stackrel{\text{def}}{=} -p\log(p) - (1-p)\log(1-p)$.

Adversarial model. By default we refer to an efficient adversary as a family of polynomial-sized circuits, or equivalently to probabilistic polynomial-time algorithm that on input of size n gets an advice string of size poly(n). However, all of our results also apply in a uniform setting in which adversaries are probabilistic polynomial-time algorithms.

2.2 Statistical and Computational indistinguishability

The *statistical distance* between discrete probability distributions X and Y is defined as $\mathrm{SD}(X,Y) \overset{\mathrm{def}}{=} \frac{1}{2}\sum_z |\Pr[X=z]-\Pr[Y=z]|$. Equivalently, the statistical distance between X and Y may be defined as the maximum, over all boolean functions T, of the *distinguishing advantage* $|\Pr[T(X)=1]-\Pr[T(Y)=1]|$. For two distribution ensembles $X=\{X_n\}$ and $Y=\{Y_n\}$, we write $X \equiv Y$ if X_n and Y_n are identically distributed, and $X \overset{\mathrm{s}}{\equiv} Y$ if the two ensembles are *statistically indistinguishable*; namely, $\mathrm{SD}(X_n,Y_n)$ is negligible in n.

A weaker notion of closeness between distributions is that of *computational* indistinguishability: We write $\{X_n\}_{n\in\mathbb{N}} \overset{\mathrm{c}}{\equiv}_{\delta(n)} \{Y_n\}_{n\in\mathbb{N}}$ if for every (non-uniform) polynomial-size circuit family $\{A_n\}$, the distinguishing advantage $|\Pr[A_n(X_n)=1]-\Pr[A_n(Y_n)=1]|$ is bounded by $\delta(n)$ for sufficiently large n. When the distinguishing advantage $\delta(n)$ is negligible, we write $\{X_n\}_{n\in\mathbb{N}} \overset{\mathrm{c}}{\equiv} \{Y_n\}_{n\in\mathbb{N}}$. (We will sometimes simplify notation and write $X_n \overset{\mathrm{c}}{\equiv} Y_n$.) By definition, $X_n \equiv Y_n$ implies that $X_n \overset{\mathrm{s}}{\equiv} Y_n$ which in turn implies that $X_n \overset{\mathrm{c}}{\equiv} Y_n$. A distribution ensemble $\{X_n\}_{n\in\mathbb{N}}$ is said to be *pseudorandom* if $X_n \overset{\mathrm{c}}{\equiv} U_{m(n)}$, where $m(n)=|X_n|$.

2.2.1 Some useful facts

We will rely on the following standard properties of statistical distance (proofs for most of these facts can be found in [SV03].)

Fact 2.2.1 *For every distributions X,Y,Z we have* $\mathrm{SD}(X,Z) \le \mathrm{SD}(X,Y)+\mathrm{SD}(Y,Z)$.

Fact 2.2.2 *For every distributions X,X',Y,Y' we have*

$$\mathrm{SD}((X\times X'),(Y\times Y')) \le \mathrm{SD}(X,Y)+\mathrm{SD}(X',Y'),$$

where $A\times B$ denotes the product distribution of A,B, i.e., the joint distribution of independent samples from A and B.

Fact 2.2.3 *For every distributions X and Y and every (possibly randomized) function A, we have* $\mathrm{SD}(A(X),A(Y)) \le \mathrm{SD}(X,Y)$.

For jointly distributed random variables A and B we write $B|_{A=a}$ to denote the conditional distribution of B given that $A=a$.

Fact 2.2.4 *Suppose that $X=(X_1,X_2)$ and $Y=(Y_1,Y_2)$ are probability distributions on a set $D\times E$ such that: (1) X_1 and Y_1 are identically distributed; and (2) with probability greater than $1-\varepsilon$ over $x\leftarrow X_1$, we have* $\mathrm{SD}(X_2|_{X_1=x},Y_2|_{Y_1=x}) \le \delta$. *Then* $\mathrm{SD}(X,Y)\le \varepsilon+\delta$.

Fact 2.2.5 *Let $\{X_z\}_{z\in\mathcal{Z}}$, $\{Y_z\}_{z\in\mathcal{Z}}$ be distribution ensembles. Then, for every distribution Z over $\mathcal{Z}$, we have* $\mathrm{SD}((Z,X_Z),(Z,Y_Z)) = \mathbb{E}_{z\leftarrow Z}[\mathrm{SD}(X_z,Y_z)]$. *In particular, if* $\mathrm{SD}(X_z,Y_z)\le\varepsilon$ *for every $z\in\mathcal{Z}$, then* $\mathrm{SD}((Z,X_Z),(Z,Y_Z))\le\varepsilon$.

We will also rely on several standard facts about computational indistinguishability (cf. [Gol01a, Chapter 2]). We begin with the computational versions of Facts 2.2.1, 2.2.2, 2.2.3.

Fact 2.2.6 *For every distribution ensembles X, Y and Z, if $X \overset{c}{\equiv} Y$ and $Y \overset{c}{\equiv} Z$ then $X \overset{c}{\equiv} Z$.*

Fact 2.2.7 *Let $\{X_n\}, \{X'_n\}, \{Y_n\}$ and $\{Y'_n\}$ be distribution ensembles. Suppose that $X_n \overset{c}{\equiv} Y_n$ and $X'_n \overset{c}{\equiv} Y'_n$. Then $(X_n \times X'_n) \overset{c}{\equiv} (Y_n \times Y'_n)$, where $A \times B$ denotes the product distribution of A, B (i.e., the joint distribution of independent samples from A and B).*

Fact 2.2.8 *Suppose that the distribution ensembles $\{X_n\}$ and $\{Y_n\}$ are computationally indistinguishable. Then for every polynomial-time computable function f we have $f(X_n) \overset{c}{\equiv} f(Y_n)$.*

Consider a case in which two probabilistic (possibly computationally unbounded) algorithms behave "similarly" on every input, in the sense that their output distributions are computationally indistinguishable. The following two facts deal with such a situation. Fact 2.2.9 asserts that an efficient procedure that gets an oracle access to one of these algorithms cannot tell which algorithm it communicates with. Fact 2.2.10 asserts that the outputs of these algorithms cannot be distinguished with respect to any (not necessarily efficiently samplable) input distribution.

Fact 2.2.9 *Let X and Y be probabilistic algorithms such that for every string family $\{z_n\}$ where $z_n \in \{0,1\}^n$, it holds that $X(z_n) \overset{c}{\equiv} Y(z_n)$. Then, for any (non-uniform) polynomial-time oracle machine A, it holds that $A^X(1^n) \overset{c}{\equiv} A^Y(1^n)$ (where A does not have access to the random coins of the given probabilistic oracle).*

Fact 2.2.10 *Let X and Y be probabilistic algorithms such that for every string family $\{z_n\}$ where $z_n \in \{0,1\}^n$, it holds that $X(z_n) \overset{c}{\equiv} Y(z_n)$. Then, for every distribution ensemble $\{Z_n\}$ where Z_n is distributed over $\{0,1\}^n$, we have $(Z_n, X(Z_n)) \overset{c}{\equiv} (Z_n, Y(Z_n))$.*

For a randomized algorithm A and an integer i we define A^i to be the randomized algorithm obtained by composing A exactly i times with itself; that is, $A^1(x) = A(x)$ and $A^i(x) = A(A^{i-1}(x))$, where in each invocation a fresh randomness is used. The following fact (which is implicit in [Ale03]) can be proved via a hybrid argument.

Fact 2.2.11 *Let $\{X_n\}$ be a distribution ensemble, and let A be a randomized polynomial-time algorithm. Suppose that $\{X_n\} \overset{c}{\equiv} \{A(X_n)\}$. Then, for every polynomial $p(\cdot)$, we have $\{X_n\} \overset{c}{\equiv} \{A^{p(n)}(X_n)\}$.*

2.3 Computational Models

Branching programs. A branching program (BP) is defined by a tuple $BP = (G, \phi, s, t)$, where $G = (V, E)$ is a directed acyclic graph, ϕ is a labeling function assigning each edge either a positive literal x_i, a negative literal $\bar{x}_i$ or the constant 1, and s, t are two distinguished nodes of G. The *size* of BP is the number of nodes in G. Each input assignment $w = (w_1, \ldots, w_n)$ naturally induces an unlabeled subgraph G_w, whose edges include all $e \in E$ such that $\phi(e)$ is satisfied by w (e.g., an edge labeled x_i is satisfied by w if $w_i = 1$). BPs may be assigned different semantics: in a *non-deterministic* BP, an input w is accepted if G_w contains at least one path from s to t; in a (*counting*) *mod-p* BP, the BP computes the number of paths from s to t modulo p. In this work, we will mostly be interested in mod-2 BPs. An example of a mod-2 BP is given in Figure 2.3.1.

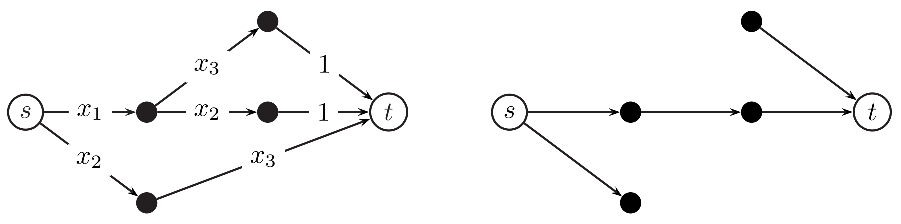

Figure 2.3.1: A mod-2 branching program computing the majority of three bits (left side), along with the graph G_{110} induced by the assignment 110 (right side).

Circuits. Boolean circuits are defined in a standard way. That is, we define a boolean circuit C as a directed acyclic graph with labeled, ordered vertices of the following types: (1) *input* vertices, each labeled with a literal x_i or $\bar{x}_i$ and having fan-in 0; (2) *gate* vertices, labeled with one of the boolean functions AND,OR and having fan-in 2; (3) *output* vertices, labeled "output" and having fan-in 1 and fan-out 0. The edges of the circuit are referred to as *wires*. A wire that outgoes from an input vertex is called an *input wire*, and a wire that enters an output vertex is called an *output wire*. Any input $x \in \{0,1\}^n$ assigns a unique *value* to each wire in the natural way. The output value of C, denoted $C(x)$, contains the values of the output wires according to the given predefined order. The *size* of a circuit, denoted $|C|$, is the number of wires in C, and its *depth* is the maximum distance from an input to an output (i.e. the length of the longest directed path in the graph).

We say that $C = \{C_n\}$ is an NC^i circuit family if for every n, the circuit C_n is of size poly(n) and depth $O(\log^i(n))$. AC^i circuits are defined similarly, except that gates are allowed to have unbounded fan-in.

NC^i-reductions. A circuit with an *oracle* access to a function $g : \{0,1\}^* \to \{0,1\}^*$ is a circuit that contains, in addition to the bounded fan-in OR, AND gates, special *oracle gates* with unbounded fan-in that compute the function g. We say that $f : \{0,1\}^* \to \{0,1\}^*$ is NC^i *reducible* to g, and write $f \in \text{NC}^i[g]$, if f can be computed by a uniform family of polynomial size, $O(\log^i n)$ depth circuits with oracle gates to g. (Oracle gates are treated the same as AND/OR gates when defining depth.) Note that if $f \in \text{NC}^i[g]$ and $g \in \text{NC}^j$ then $f \in \text{NC}^{i+j}$.

Function families and representations. We associate with a function $f : \{0,1\}^* \to \{0,1\}^*$ a function family $\{f_n\}_{n\in\mathbb{N}}$, where f_n is the restriction of f to n-bit inputs. We assume all functions to be length regular, namely their output length depends only on their input length. Hence, we may write $f_n : \{0,1\}^n \to \{0,1\}^{l(n)}$. We will represent functions f by families of circuits, branching programs, or vectors of polynomials (where each polynomial is represented by a formal sum of monomials). Whenever f is taken from a uniform class, we assume that its representation is uniform as well. That is, the representation of f_n is generated in time poly(n) and in particular is of polynomial size. We will often abuse notation and write f instead of f_n even when referring to a function on n bits. We will also write $f : \{0,1\}^n \to \{0,1\}^{l(n)}$ to denote the family $\{f_n : \{0,1\}^n \to \{0,1\}^{l(n)}\}_{n\in\mathbb{N}}$.

Complexity classes. For brevity, we use the (somewhat nonstandard) convention that all complexity classes are polynomial-time uniform unless otherwise stated. For instance, NC^0 refers to the class of functions admitting uniform NC^0 circuits, whereas *non-uniform* NC^0 refers to the class of functions admitting non-uniform NC^0 circuits. We let $\mathrm{NL}/poly$ (resp., $\oplus\mathrm{L}/poly$) denote the class of boolean functions computed by a polynomial-time uniform family of nondeterministic (resp., modulo-2) BPs. (Recall that in a uniform family of circuits or branching programs computing f, it should be possible to generate the circuit or branching program computing f_n in time $\mathrm{poly}(n)$.) Equivalently, the class $\mathrm{NL}/poly$ (resp., $\oplus\mathrm{L}/poly$) is the class of functions computed by NL (resp., $\oplus$L) Turing machines taking a uniform advice. We extend boolean complexity classes, such as $\mathrm{NL}/poly$ and $\oplus\mathrm{L}/poly$, to include non-boolean functions by letting the representation include $l(n)$ branching programs, one for each output. Uniformity requires that the $l(n)$ branching programs be all generated in time $\mathrm{poly}(n)$. Similarly, we denote by P (resp. BPP) the class of *functions* that can be computed in polynomial time (resp. probabilistic polynomial time). For instance, a function $f : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ is in BPP if there exists a probabilistic polynomial-time machine A such that for every $x \in \{0,1\}^n$ it holds that $\Pr[A(x) \neq f(x)] \leq 2^{-n}$, where the probability is taken over the internal coin tosses of A. Figure 2.3.2 describes the relations between complexity classes inside P.

$$\mathrm{NC}^0 \subsetneq \mathrm{AC}^0 \subsetneq \mathrm{TC}^0 \subseteq \mathrm{NC}^1 \subseteq \mathrm{L}/poly \subseteq \mathrm{NL}/poly, \oplus\mathrm{L}/poly \subseteq \mathrm{NC}^2 \subseteq \mathrm{NC} \subseteq \mathrm{P}$$

Figure 2.3.2: The relations between complexity classes inside P. The classes $\oplus\mathrm{L}/poly$ and $\mathrm{NL}/poly$ contain the class $\mathrm{L}/poly$ and are contained in NC^2. In a non-uniform setting the class $\oplus\mathrm{L}/poly$ contains the class $\mathrm{NL}/poly$ [Wig94].

2.4 Output locality, input locality and degree

Output Locality. Let $f : \{0,1\}^n \to \{0,1\}^l$ be a function. We say that the j-th output bit of f depends on the i-th input bit if there exists an assignment such that flipping the i-th input bit changes the value of the j-th output bit of f. The *output locality* of f is the maximal number of input bits on which an output bit of f depends. We say that the function $f : \{0,1\}^n \to \{0,1\}^l$ is *c-local* if its output locality is at most c (i.e., each of its output bits depends on at most c input bits), and that $f : \{0,1\}^* \to \{0,1\}^*$ is c-local if for every n the restriction of f to n-bit inputs is c-local.

Locality may be also defined as a syntactic property. Let C be a boolean circuit with n inputs and l outputs. Then, the j-th output wire of C depends on the i-th input wire if there exists a directed path from i to j. The *output locality* of a circuit is the maximal number of input wires on which an output wire depends. Recall that the non-uniform class nonuniform-NC^0 includes all functions $f : \{0,1\}^n \to \{0,1\}^{l(n)}$ which are computable by a circuit family $\{C_n\}$ of constant depth, polynomial size and bounded fan-in gates. For a constant c, we define the class nonuniform-NC^0_c as the class of functions which are computable by a circuit family $\{C_n\}$

of constant depth, polynomial size and bounded fan-in gates whose output locality is at most c. Clearly, the output locality of a nonuniform-NC^0 circuit is constant and therefore we can write nonuniform-$\mathrm{NC}^0 = \bigcup_{c\in\mathbb{N}}$ nonuniform-NC^0_c. The complexity classes NC^0 and NC^0_c, which are the uniform versions of the above classes, requires the circuit family $\{C_n\}$ to be polynomial-time constructible.

By definition, every function in nonuniform-NC^0_c is c-local. Also, any function in NC^0_c is both c-local and polynomial-time computable. The following proposition asserts that the converse also holds. That is, if a function is both c-local and polynomial-time computable then it is possible to efficiently construct an NC^0_c circuit that computes it.

Proposition 2.4.1 *Let $f : \{0,1\}^n \to \{0,1\}^{l(n)}$ be a c-local function which can be computed in polynomial time. Then, $f \in \mathrm{NC}^0$. That is, f can be computed by a constant depth circuit family $C = \{C_n\}$ with bounded fan-in, and there exist a polynomial-time circuit constructor A such that $A(1^n) = C_n$.*

Proof: We show that given an oracle to a c-local function f, one can efficiently "learn" an NC^0 circuit that computes f. First note that it suffices to show how to learn an NC^0 circuit for a *boolean* c-local function, as in this case we can view f as a sequence of $l(n)$ such functions $f^{(1)}, \dots, f^{(l(n))}$, one for each output bit of f, and construct an NC^0 circuit for f by concatenating the NC^0 circuits for $f^{(1)}, \dots, f^{(l(n))}$.

We now show how to learn the set S^* of variables that influence the output bit of a boolean c-local function g. Given such a procedure we can easily compute an NC^0 circuit for g (in constant time) by first recovering the truth table of g (restricted to the bits in S^*) and then converting it into a circuit via a standard transformation.

Algorithm 1 Learning the set of influencing variables of a c-local function $g : \{0,1\}^n \to \{0,1\}$.

```
1. Go over all subsets $S \subseteq [n]$ of cardinality $\leq c$, where larger sets are processed *before* smaller ones.
    (a) Let $T_S$ be the set of all strings $x \in \{0,1\}^n$ for which $x_{[n]\setminus S} = 0^{n-|S|}$.
    (b) For every input bit $i \in [n]$ test whether there exists a witness $x \in T_S$ for which $g(x) \neq g(x_{\oplus i})$. (That is, $x$ is a witness for the fact that the output of $g$ depends on the $i$-th input bit.)
    (c) If all the tests succeed, output $S$ and terminate.
```

Let S be the output of the algorithm. We prove that $S = S^*$. First note that $S \subseteq S^*$, as g depends on all the input bits of S (for each $i \in S$ we found a witness for this dependency). Hence, since Algorithm 1 processes larger sets before smaller ones, it does not terminate before it reaches S^*. It is left to show that the algorithm stops when it examines S^*. Fix some $i \in S^*$ and let $x \in \{0,1\}^n$ be the witness for the fact that g depends on i, namely, $g(x) \neq g(x_{\oplus i})$. Let $y \in \{0,1\}^n$ be a string for which $y_{S^*} = x_{S^*}$ and $y_{[n]\setminus S^*} = 0^{n-|S^*|}$. Clearly $y \in T_{S^*}$. Also, since the indices outside of S^* do not affect g, we have $g(y) = g(x') \neq g(x_{\oplus i}) = g(y_{\oplus i})$ and the proposition follows. ■

Input Locality. The *input locality* of a function f is the maximal number of output bits on which an input bit of f has influence. We envision circuits as having their inputs at the bottom and their outputs at the top. Accordingly, for functions $\ell(n), m(n)$, the class nonuniform-$\mathrm{Local}_{\ell(n)}^{m(n)}$ (resp. nonuniform-$\mathrm{Local}_{\ell(n)}$, nonuniform-$\mathrm{Local}^{m(n)}$) includes all functions $f : \{0,1\}^* \to \{0,1\}^*$ whose input locality is $\ell(n)$ and whose output locality is $m(n)$ (resp. whose input locality is $\ell(n)$, whose output locality is $m(n)$). The uniform versions of these classes contain only functions that can be computed in polynomial time. (Note that $\mathrm{Local}^{O(1)}$ is equivalent to the class NC^0. However, in most cases we will prefer the notation NC^0.) We now prove the following proposition.

Proposition 2.4.2 *A function $f : \{0,1\}^n \to \{0,1\}^{l(n)}$ is in the class $\mathrm{Local}_{O(1)}^{O(1)}$ if and only if f can be computed by polynomial-time constructible circuit family $\{C_n\}$ of constant depth, whose gates have bounded fan-in and bounded fan-out.*

Proof: Suppose that $f \in \mathrm{Local}_\ell^m$ for some constants $\ell, m \in \mathbb{N}$. Then, as shown in the proof of Proposition 2.4.1, we can construct, in time $\mathrm{poly}(n)$, an NC^0 circuit C_n that computes f_n. Moreover, the circuit computing each output bit involves only those input bits on which this output depend. Hence, the gates of C_n have also bounded fan-out.

To prove the converse direction, note that if f_n is computable by a circuit C_n with depth d, fan-in b, and fan-out c, where $b, c, d = O(1)$, then the input locality of f is at most $c^d = O(1)$ and its output locality is at most $b^d = O(1)$. Also, if C_n is constructible in time $\mathrm{poly}(n)$, then $f \in \mathrm{P}$. ■

Most of this thesis deals with output locality, hence, by default, the term *locality* always refers to *output* locality.

Degree. We will sometimes view the binary alphabet as the finite field $\mathbb{F} = \mathbb{F}_2$, and say that a function $f : \mathbb{F}^n \to \mathbb{F}^{l(n)}$ has degree d if each of its outputs can be expressed as a multivariate polynomial of degree (at most) d in the inputs. The output locality of a function trivially upper bound its degree.

Chapter 3

Randomized Encoding of Functions

Summary: In this chapter we formally introduce our notion of randomized encoding which will be used as a central tool in subsequent chapters. In Section 3.1 we introduce several variants of randomized encoding and in Section 3.2 we prove some of their useful properties. Finally, in Section 3.3 we discuss other aspects and limitations of randomized encoding such as the necessity of randomness, the expressive power of encoding in NC^0, and the lowest output locality which is sufficient to encode all functions.

3.1 Definitions

We start by defining a randomized encoding of a finite function f. This definition will be later extended to a (uniform) family of functions.

Definition 3.1.1 (Randomized encoding) *Let $f : \{0,1\}^n \to \{0,1\}^l$ be a function. We say that a function $\hat{f} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$ is a δ-correct, ε-private* randomized encoding *of f, if it satisfies the following:*

- **δ-correctness.** *There exists a deterministic*[1] *algorithm B, called a* decoder*, such that for every input $x \in \{0,1\}^n$, $\Pr[B(\hat{f}(x, U_m)) \neq f(x)] \leq \delta$.*

- **ε-privacy.** *There exists a randomized algorithm S, called a* simulator*, such that for every $x \in \{0,1\}^n$, $\mathrm{SD}(S(f(x)), \hat{f}(x, U_m)) \leq \varepsilon$.*

We refer to the second input of $\hat{f}$ as its random input *and to m and s as the* randomness complexity *and* output complexity *of $\hat{f}$, respectively.*

Note that the above definition only refers to the *information* about x revealed by $\hat{f}(x, r)$ and does not consider the complexity of the decoder and the simulator. Intuitively, the function $\hat{f}$ defines an "information-theoretically equivalent" representation of f. The correctness property guarantees that from $\hat{y} = \hat{f}(x, r)$ it is possible to reconstruct $f(x)$ (with high probability), whereas the privacy property guarantees that by seeing $\hat{y}$ one cannot learn too much about x (in addition to $f(x)$). The encoding is δ-correct (resp. ε-private), if it correct (resp. private) up to an "error" of δ (resp., ε). This is illustrated by the next example.

[1]We restrict the decoder to be deterministic for simplicity. This restriction does not compromise generality, in the sense that one can transform a randomized decoder to a deterministic one by incorporating the coins of the former in the encoding itself.

Example 3.1.2 Consider the function $f(x_1,\ldots,x_n) = x_1 \vee x_2 \vee \ldots \vee x_n$. We define a randomized encoding $\hat{f} : \{0,1\}^n \times \{0,1\}^{ns} \to \{0,1\}^s$ by $\hat{f}(x,r) = (\sum_{i=1}^n x_i r_{i,1}, \ldots, \sum_{i=1}^n x_i r_{i,s})$, where $x = (x_1,\ldots,x_n)$, $r = (r_{i,j})$ for $1 \leq i \leq n, 1 \leq j \leq s$, and addition is over $\mathbb{F}_2$. First, observe that the distribution of $\hat{f}(x, U_{ns})$ depends only on the value of $f(x)$. Specifically, let S be a simulator that outputs an s-tuple of zeroes if $f(x) = 0$, and a uniformly chosen string in $\{0,1\}^s$ if $f(x) = 1$. It is easy to verify that $S(f(x))$ is distributed the same as $\hat{f}(x, U_{ns})$ for any $x \in \{0,1\}^n$. It follows that this randomized encoding is 0-private. Also, one can obtain an efficient decoder B that given a sample y from the distribution $\hat{f}(x, U_{ns})$ outputs 0 if $y = 0^s$ and otherwise outputs 1. Such an algorithm will err with probability 2^{-s}, thus $\hat{f}$ is 2^{-s}-correct.

On uniform randomized encodings. The above definition naturally extends to functions $f : \{0,1\}^* \to \{0,1\}^*$. In this case, the parameters $l, m, s, \delta, \varepsilon$ are all viewed as functions of the input length n, and the algorithms B, S receive 1^n as an additional input. In our default uniform setting, we require that $\hat{f}_n$, the encoding of f_n, be computable in time poly(n) (given $x \in \{0,1\}^n$ and $r \in \{0,1\}^{m(n)}$). Thus, in this setting both $m(n)$ and $s(n)$ are polynomially bounded. We also require both the decoder and the simulator to be efficient. (This is not needed by some of the applications, but is a feature of our constructions.) We formalize these requirements below.

Definition 3.1.3 (Uniform randomized encoding) *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function and $l(n)$ an output length function such that $|f(x)| = l(|x|)$ for every $x \in \{0,1\}^*$. We say that $\hat{f} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ is a $\delta(n)$-correct $\epsilon(n)$-private uniform randomized encoding of f, if the following holds:*

- **Length regularity.** *There exist polynomially-bounded and efficiently computable length functions $m(n), s(n)$ such that for every $x \in \{0,1\}^n$ and $r \in \{0,1\}^{m(n)}$, we have $|\hat{f}(x,r)| = s(n)$.*
- **Efficient evaluation.** *There exists a polynomial-time* evaluation algorithm *that, given $x \in \{0,1\}^*$ and $r \in \{0,1\}^{m(|x|)}$, outputs $\hat{f}(x,r)$.*
- **δ-correctness.** *There exists a polynomial-time* decoder *B, such that for every $x \in \{0,1\}^n$ we have* $\Pr[B(1^n,\ \hat{f}(x, U_{m(n)})) \neq f(x)] \leq \delta(n)$.
- **ε-privacy.** *There exists a probabilistic polynomial-time* simulator *S, such that for every $x \in \{0,1\}^n$ we have* $\mathrm{SD}(S(1^n, f(x)), \hat{f}(x, U_{m(n)})) \leq \varepsilon(n)$.

When saying that a uniform encoding $\hat{f}$ is in a (uniform) circuit complexity class, we mean that its evaluation algorithm can be implemented by circuits in this class. For instance, we say that $\hat{f}$ is in NC^0_d if there exists a polynomial-time circuit generator G such that $G(1^n)$ outputs a d-local circuit computing $\hat{f}(x,r)$ on all $x \in \{0,1\}^n$ and $r \in \{0,1\}^{m(n)}$.

From here on, a randomized encoding of an efficiently computable function is assumed to be uniform by default. Moreover, we will freely extend the above definition to apply to a uniform collection of functions $\mathcal{F} = \{f_z\}_{z \in Z}$, for some index set $Z \subseteq \{0,1\}^*$. In such a case it is required that the encoded collection $\hat{\mathcal{F}} = \{\hat{f}_z\}_{z \in Z}$ is also uniform, in the sense that the same efficient evaluation algorithm, decoder, and simulator should apply to the entire collection when given z as an additional input. (See Appendix A for a more detailed discussion of *collections* of functions and cryptographic primitives.) Finally, for the sake of simplicity we will sometimes formulate our definitions, claims and proofs using finite functions, under the implicit understanding that they naturally extend to the uniform setting.

We move on to discuss some variants of the basic definition. Correctness (resp., privacy) can be either *perfect*, when $\delta = 0$ (resp., $\varepsilon = 0$), or *statistical*, when $\delta(n)$ (resp., $\varepsilon(n)$) is negligible. In fact, we can further relax privacy to hold only against efficient adversaries, e.g., to require that for every $x \in \{0,1\}^n$, every polynomial-size circuit family $\{A_n\}$ distinguishes between the distributions $S(f(x))$ and $\hat{f}(x, U_m)$ with no more than negligible advantage. Such an encoding is referred to as *computationally* private and it suffices for the purpose of many applications. However, while for some of the primitives (such as OWF) computational privacy and statistical correctness will do, others (such as PRGs or one-way permutations) require even stronger properties than perfect correctness and privacy. One such additional property is that the simulator S, when invoked on a uniformly random string from $\{0,1\}^l$ (the output domain of f), will output a uniformly random string from $\{0,1\}^s$ (the output domain of $\hat{f}$). We call this property *balance.* Note that the balance requirement does not impose any uniformity condition on the output of f, which in fact can be concentrated on a strict subset of $\{0,1\}^l$.

Definition 3.1.4 (Balanced randomized encoding) *A randomized encoding $\hat{f} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$ of a function $f : \{0,1\}^n \to \{0,1\}^l$ is called* balanced *if it has a perfectly private simulator S such that $S(U_l) \equiv U_s$. We refer to S as a* balanced simulator.

A last useful property is a syntactic one: we sometimes want $\hat{f}$ to have the same additive stretch as f. Specifically, we say that $\hat{f}$ is *stretch-preserving* (with respect to f) if $s - (n + m) = l - n$, or equivalently $m = s - l$.

We are now ready to define our three main variants of randomized encoding.

Definition 3.1.5 (Perfect randomized encoding) *A* perfect randomized encoding *is a randomized encoding that is perfectly correct, perfectly private, balanced, and stretch-preserving.*

Definition 3.1.6 (Statistical randomized encoding) *A* statistical randomized encoding *is a randomized encoding that is statistically correct and statistically private.*

Definition 3.1.7 (Computational randomized encoding) *Let $f = \{f_n : \{0,1\}^n \to \{0,1\}^{\ell(n)}\}$ be a function family. We say that the function family $\hat{f} = \{\hat{f}_n : \{0,1\}^n \times \{0,1\}^{m(n)} \to \{0,1\}^{s(n)}\}$ is a* computational randomized encoding *of f (or computational encoding for short), if it satisfies the following requirements:*

- **Statistical correctness.** *There exists a polynomial-time decoder B, such that for every $x \in \{0,1\}^n$, we have $\Pr[B(1^n, \hat{f}_n(x, U_{m(n)})) \neq f_n(x)] \leq \delta(n)$, for some negligible function $\delta(n)$.*

- **Computational privacy.** *There exists a probabilistic polynomial-time simulator S, such that for any family of strings $\{x_n\}_{n\in\mathbb{N}}$ where $|x_n| = n$, we have $S(1^n, f_n(x_n)) \overset{c}{\equiv} \hat{f}_n(x_n, U_{m(n)})$.*

We will also refer to *perfectly correct* computational encodings, where the statistical correctness requirement is strengthened to perfect correctness. (In fact, the construction of Section 5.2 yields such an encoding.)

A combinatorial view of perfect encoding. To gain better understanding of the properties of perfect encoding, we take a closer look at the relation between a function and its encoding. Let $\hat{f} : \{0,1\}^{n+m} \to \{0,1\}^s$ be an encoding of $f : \{0,1\}^n \to \{0,1\}^l$. The following description addresses the simpler case where f is onto. Every $x \in \{0,1\}^n$ is mapped to some $y \in \{0,1\}^l$ by f, and to a 2^m-size multiset $\{\hat{f}(x,r) | r \in \{0,1\}^m\}$ which is contained in $\{0,1\}^s$. Perfect privacy means that this multiset is common to all the x's that share the same image under f; so we have a mapping from $y \in \{0,1\}^l$ to multisets in $\{0,1\}^s$ of size 2^m (such a mapping is defined by the perfect simulator). Perfect correctness means that these multisets are mutually disjoint. However, even perfect privacy and perfect correctness together do not promise that this mapping covers all of $\{0,1\}^s$. The balance property guarantees that the multisets form a perfect tiling of $\{0,1\}^s$; moreover it promises that each element in these multisets has the same multiplicity. If the encoding is also stretch-preserving, then the multiplicity of each element must be 1, so that the multisets are actually sets. Hence, a perfect randomized encoding guarantees the existence of a perfect simulator S whose 2^l output distributions form a perfect tiling of the space $\{0,1\}^s$ by sets of size 2^m.

Remark 3.1.8 (A padding convention) We will sometimes view $\hat{f}$ as a function of a single input of length $n + m(n)$ (e.g., when using it as a OWF or a PRG). In this case, we require $m(\cdot)$ to be monotone non-decreasing, so that $n + m(n)$ uniquely determines n. We apply a standard padding technique for defining $\hat{f}$ on inputs whose length is not of the form $n + m(n)$. Specifically, if $n + m(n) + t < (n+1) + m(n+1)$ we define $\hat{f}'$ on inputs of length $n + m(n) + t$ by applying $\hat{f}_n$ on the first $n + m(n)$ bits and then appending the t additional input bits to the output of $\hat{f}_n$. This convention respects the security of cryptographic primitives such as OWF, PRG, and collision-resistant hashing, provided that $m(n)$ is efficiently computable and is sufficiently dense (both of which are guaranteed by a uniform encoding). That is, if the unpadded function $\hat{f}$ is secure with respect to its partial domain, then its padded version $\hat{f}'$ is secure in the standard sense, i.e., over the domain of all strings.[2] (See a proof for the case of OWF in [Gol01a, Proposition 2.2.3].) Note that the padded function $\hat{f}'$ has the same locality and degree as $\hat{f}$. Moreover, $\hat{f}'$ also preserves syntactic properties of $\hat{f}$; for example it preserves the stretch of $\hat{f}$, and if $\hat{f}$ is a permutation then so is $\hat{f}'$. Thus, it is enough to prove our results for the partially defined unpadded function $\hat{f}$, and keep the above conventions implicit.

Finally, we define three complexity classes that capture the power of randomized encodings in NC^0.

Definition 3.1.9 (The classes PREN, SREN, CREN) *The class $\mathcal{PREN}$ (resp., $\mathcal{SREN}$, $\mathcal{CREN}$) is the class of functions $f : \{0,1\}^* \to \{0,1\}^*$ admitting a perfect (resp., statistical, computational) uniform randomized encoding in* NC^0. *(As usual,* NC^0 *is polynomial-time uniform.)*

It follows from the definitions that $\mathcal{PREN} \subseteq \mathcal{SREN} \subseteq \mathcal{CREN}$.[3] We will later show that $\oplus\mathrm{L}/poly \subseteq \mathcal{PREN}$ (Theorem 4.2.6), $\mathrm{NL}/poly \subseteq \mathcal{SREN}$ (Theorem 4.2.8), and, assuming that there exists a PRG in $\mathcal{PREN}$, the class $\mathcal{CREN}$ is equal to the class BPP (Theorem 5.2.15).

[2]This can be generally explained by viewing each slice of the padded function $\hat{f}'$ (i.e., its restriction to inputs of some fixed length) as a *perfect* randomized encoding of a corresponding slice of $\hat{f}$.

[3]We also note that if $\mathcal{CREN} \neq \mathcal{SREN}$ then there exist two polynomial-time constructible ensembles which are computationally indistinguishable but not statistically close. In [Gol90] it is shown that such ensembles implies the existence of infinitely often OWF, i.e., a polynomial-time computable function which is hard to invert for infinitely many input lengths (see [Gol01a, Def. 4.5.4] for formal definition).

Moreover, functions in all of these classes admit an encoding of degree 3 and (output) locality 4 (Corollary 4.2.9).

3.2 Basic Properties

We now put forward some useful properties of randomized encodings. We first argue that an encoding of a non-boolean function can be obtained by concatenating encodings of its output bits, using an independent random input for each bit. The resulting encoding inherits all the features of the concatenated encodings, and in particular preserves their perfectness.

Lemma 3.2.1 (Concatenation) *Let $f_i : \{0,1\}^n \to \{0,1\}$, $1 \le i \le l$, be the boolean functions computing the output bits of a function $f : \{0,1\}^n \to \{0,1\}^l$. If $\hat{f}_i : \{0,1\}^n \times \{0,1\}^{m_i} \to \{0,1\}^{s_i}$ is a δ-correct ε-private encoding of f_i, then the function $\hat{f} : \{0,1\}^n \times \{0,1\}^{m_1+\ldots+m_l} \to \{0,1\}^{s_1+\ldots+s_l}$ defined by $\hat{f}(x,(r_1,\ldots,r_l)) \overset{def}{=} (\hat{f}_1(x,r_1),\ldots,\hat{f}_l(x,r_l))$ is a (δl)-correct, (εl)-private encoding of f. Moreover, if all $\hat{f}_i$ are perfect then so is $\hat{f}$.*

Proof: We start with correctness. Let B_i be a δ-correct decoder for $\hat{f}_i$. Define a decoder B for $\hat{f}$ by $B(\hat{y}_1,\ldots,\hat{y}_l) = (B_1(\hat{y}_1),\ldots,B_l(\hat{y}_l))$. By a union bound argument, B is a (δl)-correct decoder for $\hat{f}$ as required.

We turn to analyze privacy. Let S_i be an ε-private simulator for $\hat{f}_i$. An (εl)-private simulator S for $\hat{f}$ can be naturally defined by $S(y) = (S_1(y_1),\ldots,S_l(y_l))$, where the invocations of the simulators S_i use independent coins. Indeed, for every $x \in \{0,1\}^n$ we have:

$$\begin{aligned}
\mathrm{SD}(S(f(x)),\hat{f}(x,(U_{m_1},\ldots,U_{m_l}))) &= \mathrm{SD}((S_1(y_1),\ldots,S_l(y_l)),(\hat{f}_1(x,U_{m_1}),\ldots,\hat{f}_l(x,U_{m_l}))) \\
&\le \sum_{i=1}^{l} \mathrm{SD}(S_i(y_i),\hat{f}_i(x,U_{m_i})) \\
&\le \varepsilon l,
\end{aligned}$$

where $y = f(x)$. The first inequality follows from Fact 2.2.2 and the independence of the randomness used for different i, and the second from the ε-privacy of each S_i.

Note that the simulator S described above is balanced if all S_i are balanced. Moreover, if all $\hat{f}_i$ are stretch preserving, i.e., $s_i - 1 = m_i$, then we have $\sum_{i=1}^{l} s_i - l = \sum_{i=1}^{l} m_i$ and hence $\hat{f}$ is also stretch preserving. It follows that if all $\hat{f}_i$ are perfect then so is $\hat{f}$. ■

We state the following uniform version of Lemma 3.2.1, whose proof is implicit in the above.

Lemma 3.2.2 (Concatenation: uniform version) *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function, viewed as a uniform collection of functions $\mathcal{F} = \{f_{n,i}\}_{n\in\mathbb{N},1\le i\le l(n)}$; that is, $f_{n,i}(x)$ outputs the i-th bit of $f(x)$ for all $x \in \{0,1\}^n$. Suppose that $\hat{\mathcal{F}} = \{\hat{f}_{n,i}\}_{n\in\mathbb{N},1\le i\le l(n)}$ is a perfect (resp., statistical) uniform randomized encoding of $\mathcal{F}$. Then, the function $\hat{f} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ defined by $\hat{f}(x,(r_1,\ldots,r_{l(|x|)})) \overset{def}{=} (\hat{f}_{|x|,1}(x,r_1),\ldots,\hat{f}_{|x|,l(|x|)}(x,r_{l(|x|)}))$ is a perfect (resp., statistical) uniform randomized encoding of f.*

Another useful feature of randomized encodings is the following intuitive composition property: suppose we encode f by g, and then view g as a deterministic function and encode it again. Then, the resulting function (parsed appropriately) is a randomized encoding of f. Again, the resulting encoding inherits the perfectness of the encodings from which it is composed.

Lemma 3.2.3 (Composition) *Let $g(x, r_g)$ be a δ_g-correct, ε_g-private encoding of $f(x)$ and $h((x, r_g), r_h)$ be a δ_h-correct, ε_h-private encoding of $g((x, r_g))$ (viewed as a single-argument function). Then, the function $\hat{f}(x,(r_g, r_h)) \stackrel{def}{=} h((x, r_g), r_h)$ is a $(\delta_g + \delta_h)$-correct, $(\varepsilon_g + \varepsilon_h)$-private encoding of f. Moreover, if g, h are perfect (resp., statistical) uniform randomized encodings then so is $\hat{f}$.*

Proof: We start with correctness. Let B_g be a δ_g-correct decoder for g and B_h a δ_h-correct decoder for h. Define a decoder B for $\hat{f}$ by $B(\hat{y}) = B_g(B_h(\hat{y}))$. The decoder B errs only if either B_h or B_g err. Thus, by the union bound we have for every x,

$$\begin{aligned}\Pr_{r_g,r_h}[B(\hat{f}(x,(r_g,r_h))) \neq f(x)] &\leq \Pr_{r_g,r_h}[B_h(h((x,r_g),r_h)) \neq g(x,r_g)] \\ &\quad + \Pr_{r_g}[B_g(g(x,r_g)) \neq f(x)] \\ &\leq \delta_h + \delta_g,\end{aligned}$$

as required.

Privacy is argued similarly. Let S_g be an ε_g-private simulator for g and S_h an ε_h-private simulator for h. We define a simulator S for $\hat{f}$ by $S(y) = S_h(S_g(y))$. Letting m_g, m_h denote the randomness complexity of g, h, respectively, we have for every x,

$$\begin{aligned}\mathrm{SD}(S(f(x)), \hat{f}(x,(U_{m_g}, U_{m_h}))) &= \mathrm{SD}(S_h(S_g(f(x))), h((x, U_{m_g}), U_{m_h})) \\ &\leq \mathrm{SD}(S_h(S_g(f(x))), S_h(g(x, U_{m_g}))) \\ &\quad + \mathrm{SD}(S_h(g(x, U_{m_h})), h((x, U_{m_g}), U_{m_h})) \\ &\leq \varepsilon_g + \varepsilon_h,\end{aligned}$$

where the first inequality follows from the triangle inequality (Fact 2.2.1), and the second from Facts 2.2.3 and 2.2.5.

It is easy to verify that if S_g and S_h are balanced then so is S. Moreover, if g preserves the additive stretch of f and h preserves the additive stretch of g then h (hence also $\hat{f}$) preserves the additive stretch of f. Thus $\hat{f}$ is perfect if both g, h are perfect. All the above naturally carries over to the uniform setting, from which the last part of the lemma follows. ■

The composition lemma can be easily extended to the computational setting.

Lemma 3.2.4 (Composition of computational encoding) *Let $g(x, r_g)$ be a computational encoding of $f(x)$ and $h((x, r_g), r_h)$ a computational encoding of $g((x, r_g))$, viewing the latter as a single-argument function. Then, the function $\hat{f}(x,(r_g, r_h)) \stackrel{def}{=} h((x, r_g), r_h)$ is a computational encoding of $f(x)$ whose random inputs are (r_g, r_h). Moreover, if g, h are perfectly correct then so is $\hat{f}$.*

Proof: Correctness follows from the same arguments as in the proof of Lemma 3.2.3. To prove computational privacy, we again define a simulator S for $\hat{f}$ by $S(y) = S_h(S_g(y))$, where S_g (resp., S_h) is a computationally-private simulator for g (resp., h). Letting $m_g(n)$ and $m_h(n)$ denote the randomness complexity of g and h, respectively, and $\{x_n\}_{n\in\mathbb{N}}$ be a family of strings where $|x_n| = n$, we have,

$$\begin{aligned} S_h(S_g(f(x_n))) &\overset{c}{\equiv} S_h(g(x_n, U_{m_g(n)})) && \text{(comp. privacy of } g\text{, Fact 2.2.8)} \\ &\overset{c}{\equiv} h((x_n, U_{m_g(n)}), U_{m_h(n)}) && \text{(comp. privacy of } h\text{, Fact 2.2.10).} \end{aligned}$$

Hence, the transitivity of the relation $\overset{c}{\equiv}$ (Fact 2.2.6) completes the proof. ■

It follows as a special case that the composition of a computational encoding with a perfect or a statistical encoding is a computational encoding.

Finally, we prove two useful features of a *perfect* encoding.

Lemma 3.2.5 (Unique randomness) *Suppose $\hat{f}$ is a perfect randomized encoding of f. Then, (a) $\hat{f}$ satisfies the following* unique randomness *property: for any input x, the function $\hat{f}(x,\cdot)$ is injective, namely there are no distinct r, r' such that $\hat{f}(x,r) = \hat{f}(x,r')$. Moreover, (b) if f is a permutation then so is $\hat{f}$.*

Proof: Let $f : \{0,1\}^n \to \{0,1\}^l$ and $\hat{f} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$. To prove part (a), assume towards a contradiction that $\hat{f}$ does not satisfy the unique randomness property. Then, by perfect privacy, we have $|\mathrm{Im}(\hat{f})| < |\mathrm{Im}(f)| \cdot 2^m$. On the other hand, letting S be a balanced simulator, we have

$$\begin{aligned} |\mathrm{Im}(\hat{f})| \cdot 2^{-s} &= \Pr_{y\leftarrow U_l}[S(y) \in \mathrm{Im}(\hat{f})] \\ &\geq \Pr_{y\leftarrow U_l}[S(y) \in \mathrm{Im}(\hat{f}) | y \in \mathrm{Im}(f)] \cdot \Pr_{y\leftarrow U_l}[y \in \mathrm{Im}(f)] \\ &= 1 \cdot \frac{|\mathrm{Im}(f)|}{2^l}, \end{aligned}$$

where the last equality follows from perfect privacy. Since g is stretch preserving ($s - l = m$), we get from the above that $|\mathrm{Im}(\hat{f})| \geq |\mathrm{Im}(f)| \cdot 2^m$, and derive a contradiction.

If f is a permutation then $n = l$ and since $\hat{f}$ is stretch preserving, we can write $\hat{f} : \{0,1\}^s \to \{0,1\}^s$. Thus, to prove part (b), it is enough to prove that $\hat{f}$ is injective. Suppose that $\hat{f}(x,r) = \hat{f}(x',r')$. Then, since f is injective and $\hat{f}$ is perfectly correct it follows that $x = x'$; hence, by part (a), $r = r'$ and the proof follows. ■

3.3 More Aspects of Randomized Encoding

In the followings we address some natural questions regarding the complexity of randomized encoding.

3.3.1 The Necessity of Randomness

We begin by asking whether randomization is really needed in order to encode a function f by a function $\hat{f}$ with constant output locality (i.e., NC^0 function) or constant input locality. The following observation shows that, at least when f is a boolean function, only trivial functions admit such an encoding.

Observation 3.3.1 *If $f : \{0,1\}^n \to \{0,1\}$ can be encoded by a function $\hat{f} : \{0,1\}^n \to \{0,1\}^s$ whose output locality (resp. input locality) is c, then the output locality (resp. input locality) of f is c.*

Proof: The privacy of the encoding promises that there exists a pair of strings $y_0, y_1 \in \{0,1\}^s$ such that for every $x \in \{0,1\}^n$ we have $\hat{f}(x) = y_{f(x)}$. Also, by perfect correctness, $y_0 \neq y_1$. Assume, without loss of generality, that y_0 and y_1 differ in the first bit. Then, we can compute $f(x)$ by computing the first bit of $\hat{f}(x)$ or its negation. Thus, the output locality and input locality of f are equal to those of $\hat{f}$. ∎

The above argument relies heavily on the fact that f is a boolean function. Indeed, the claim does not hold in the case of non-boolean functions. Suppose, for example, that $f : \{0,1\}^n \to \{0,1\}^n$ is a permutation. Then it can be trivially encoded in Local_1^1 by the identity function. Moreover, if f can be computed and inverted in polynomial time, then the encoding allows efficient decoding and simulation.

3.3.2 The Power of Randomized Encoding

Statistical encoding for functions outside of NC

It is very interesting to find out what is the power of randomized encoding in NC^0; that is, which functions are in $\mathcal{SREN}$. Since our general constructions of statistical randomized encoding apply only to various logspace classes (e.g., $\mathrm{NL}/\mathit{poly}, \oplus\mathrm{L}/\mathit{poly}$), one might suspect that $\mathcal{SREN}$ is limited to such functions. It turns out that there are functions in $\mathcal{SREN}$ that are not known to be computable even in NC. Consider, for example, the collection of boolean functions $\{QR_p(x)\}$, which check whether x is a quadratic residue modulo a prime p (i.e., this collection is indexed by primes). This collection can be encoded by the collection $\hat{QR}_p(x,r) = xr^2 \mod p$ where r is uniformly chosen from $[p]$. Since $\{\hat{QR}_p\}$ can be approximated (up to a negligible error) by a family of NC^1 circuits (over binary alphabet), we can statistically encode it by an NC^0 collection, which, by composition (Lemma 3.2.3), also encodes the collection $\{QR_p\}$. However, the collection $\{QR_p\}$ is not known to be computable in NC.

Randomized Encoding with Unbounded Decoder and Unbounded Simulator

For some applications, we can relax the uniformity of randomized encoding and allow the decoder and/or the simulator to be computationally unbounded.[4] For example, a perfect encoding with an unbounded simulator and an unbounded decoder still preserves the security of collision resistent hashing. Moreover, the security of several primitives is preserved by an encoding with an efficient simulator and an unbounded decoder (see Table 4.9.1). Such variants are also useful for information theoretic secure computation between computationally unbounded parties (cf. [BMR90]). Hence, we would like to understand the power of randomized encodings in this setting too.

Of course, whenever the decoder of the encoding is restricted to run in polynomial time we cannot hope to represent functions that are not efficiently computable (i.e., out of BPP). However, it turns out that if we do not restrict the running time of the decoder, some functions that are assumed to be intractable can be encoded by NC^0 functions. For example, consider the function GI

[4]The encoding itself should still be computable in (polynomial-time) uniform NC^0.

that given a graph G (represented as an $n \times n$ adjacency matrix), outputs the lexicographically first graph H that is isomorphic to G. This function is not known to be computable by a polynomial time algorithm (as such an algorithm would imply that the graph isomorphism language is in P). However, we can encode GI by the function $\hat{GI}(G, r) = M(r)GM^T(r)$ where $M(\cdot)$ is a mapping from random bits to (almost) uniformly chosen $n \times n$ permutation matrix. It is not hard to verify that $\hat{GI}$ is a statistical encoding of GI. Also, the encoding $\hat{GI}$ can be computed in NC^1, and therefore GI can be encoded by an NC^0 encoding (with an efficient simulator). A similar example (for a function that is not known to be efficiently computable but can be represented by an NC^0 encoding) can be obtained by a variant of the quadratic residuosity function presented above in which the modulus is a composite.

On the other hand, the following observation shows that "hard" functions are unlikely to have efficiently computable encodings, even if we allow non-efficient simulation and decoding.

Observation 3.3.2 *Let f be a boolean function, L be the language that corresponds to f (i.e., $L = f^{-1}(1)$) and $\hat{f}$ be a polynomial time computable encoding of f. Then,*

1. *If the encoding is perfect and the simulator is efficient then $L \in \mathrm{NP} \cap \mathrm{co}-\mathrm{NP}$.*

2. *If the encoding is perfect (and the simulator is not necessarily efficient) then $L \in$ (non-uniform) $\mathrm{NP}/poly \cap$ (non-uniform) $\mathrm{co}-\mathrm{NP}/poly$.*

3. *If the encoding is statistical and the simulator is efficient then L has a statistical zero-knowledge proof system.*

Proof sketch:

1. We use the randomness of the encoding as a witness; namely, $f(x) = b$ iff $\exists r$ such that $S_{\bar{0}}(b) = \hat{f}(x, r)$ (i.e., we fix the random coins of the simulator to be the all-zero string.)

2. We use the same argument as in (1), but instead of computing $S_{\bar{0}}(b)$ we use the advice $\hat{f}(x_b, \bar{0})$, where $\bar{0}$ is the all-zero string and x_b is a fixed n-bit string that satisfies $f(x_b) = b$.

3. The protocol is very similar to the standard ZK protocol of graph non-isomorphism [GMR89]; namely, the verifier selects randomly $b \in \{0, 1\}$: if $b = 1$ it sends a sample from $\hat{f}(x, r)$, otherwise it sends a sample from $S(0)$. The prover has to find the bit b; if the prover succeeds then the verifier accepts, otherwise the verifier rejects. To see that this is indeed a ZK-protocol (against an honest-verifier), note that $f(x) = 1$ implies that the distributions $\hat{f}(x, r)$ and $S(0)$ are almost disjoint, and $f(x) = 0$ implies that these distributions are statistically close.[5]

■

As a corollary we can deduce that in all the above settings one is unlikely to obtain a polynomial time computable encoding for an NP-hard language.

[5]In fact, this is a specific instance of the statistical difference problem which was shown to be complete for the class SZK[SV03].

3.3.3 Lower Bounds for Locality and Degree

In Section 4.2 we will show that every function f can be perfectly encoded by a degree 3 encoding in NC_4^0 whose complexity is polynomial in the size of the branching program that computes f. In [IK00, Corollary 5.9] it was shown that most boolean functions do not admit perfectly-private randomized encoding of degree 2 regardless of the efficiency of the encoding. (In fact, an exact characterization of the class of functions that admit such an encoding is given.) Hence, 3 is the minimal degree which is sufficient to (perfectly) encode all functions. We now show that locality 4 is also minimal with respect to perfect encoding.

We say that a function $f : \{0,1\}^n \to \{0,1\}^l$ is ε-balanced if $\mathrm{SD}(f(U_n), U_l) \leq \varepsilon$. When $\varepsilon = 0$ we will say that f is balanced. The following claim shows that if a c-local function is $\frac{1}{2} \cdot 2^{-c}$-balanced then it is actually balanced and its degree is bounded by $c - 1$.

Claim 3.3.3 *Let $f : \{0,1\}^c \to \{0,1\}$ be a $\frac{1}{2} \cdot 2^{-c}$-balanced function. Then, f is balanced and its degree is at most $c - 1$.*

Proof: In the following we will identify the truth table of a function $g : \{0,1\}^c \to \{0,1\}$ with a string $z \in \{0,1\}^{2^c}$ which is indexed by c-bit strings such that $z_x = g(x)$.

First note that when f is not balanced its bias $\left|\Pr_x[f(x) = 1] - \frac{1}{2}\right|$ is at least 2^{-c}. Hence, in this case $\mathrm{SD}(f(U_n), U_1) \geq 2^{-c}$ and so it cannot be $\frac{1}{2} \cdot 2^{-c}$-balanced. We conclude that f is balanced.

Now assume, towards a contradiction, that f is a function of degree exactly c. Then we can write f as a sum of monomials $f(x) = T_1(x) + \ldots + T_k(x)$ where $T_1(x) = x_1 \cdot \ldots \cdot x_c$ and $T_2, \ldots, T_k$ are monomials of degree $\leq c - 1$. Let z be the truth table of f, and let z_i be the truth table of T_i. The following three claims show that the truth table z of f has an odd number of ones and thus f cannot be balanced: (1) $z = z_1 \oplus \ldots \oplus z_k$; (2) the number of ones in z_1 is odd; and (3) for $1 < i \leq k$, the number of ones in z_i is even. The first two claims can be easily verified, and claim (3) follows from the fact that each T_i depends on a strict subset of the inputs. ■

We also need the following claim.

Claim 3.3.4 *Let $f : \{0,1\}^n \to \{0,1\}^l$ be an ε-balanced function which is perfectly encoded by $\hat{f} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$. Then, $\hat{f}$ is also ε-balanced.*

Proof: Let S be the balanced simulator of $\hat{f}$. Then,

$$\begin{aligned}
\mathrm{SD}(\hat{f}(U_n, U_m), U_s) &\leq \mathrm{SD}(\hat{f}(U_n, U_m), S(f(U_n))) + \mathrm{SD}(S(f(U_n)), S(U_l)) + \mathrm{SD}(S(U_l), U_s) \\
&= 0 + \mathrm{SD}(S(f(U_n)), S(U_l)) + 0 \\
&\leq \mathrm{SD}(f(U_n), U_l) \leq \varepsilon,
\end{aligned}$$

where the first inequality follows from Fact 2.2.1, the first equality is due to perfect privacy, Fact 2.2.5 and the fact that S is a balanced simulator, the second inequality follows from Fact 2.2.3, and the last inequality is due to the fact that f is ε-balanced. ■

By combining the above claims and the results of [IK00, Corollary 5.9], we get the following corollary:

Corollary 3.3.5 *For sufficiently large n, most boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}$ cannot be perfectly encoded by functions $\hat{f} \in \mathrm{NC}^0_3$.*

Proof: We begin by proving the claim for a function f which is 2^{-4}-balanced and cannot be perfectly encoded by a degree 2 encoding. Assume, towards a contradiction, that f is perfectly encoded by a 3-local function $\hat{f}$. Then, by Claim 3.3.4, $\hat{f}$ is also 2^{-4}-balanced and thus, by Claim 3.3.3, $\hat{f}$ is a degree-2 encoding and we derive a contradiction.

We complete the proof by noting that most functions satisfy the above properties. Indeed, for sufficiently large n, all but a negligible fraction of the boolean functions over n-bits are 2^{-4}-balanced. (This can be proved by choosing a random function and applying a Chernoff bound.) Also, by [IK00, Corollary 5.9], only negligible fraction of the boolean functions can be perfectly encoded by degree 2 functions. ■

In particular, it follows from [IK00, Corollary 5.9] that the function $x_1 \cdot x_2 \cdot x_3 + x_4$ cannot be perfectly encoded by a degree 2 function, hence, by Corollary 3.3.5, this function cannot be perfectly encoded by a 3-local function (regardless of the complexity of the encoding).

Chapter 4

Cryptography in NC^0

Summary: In this chapter we show that randomized encoding preserves the security of many cryptographic primitives. We also construct an (information-theoretic) encoding in NC^0_4 for any function in NC^1 or even $\oplus\mathrm{L}/poly$. The combination of these results gives a compiler that takes as an input a code of an NC^1 implementation of some cryptographic primitive and generates an NC^0_4 implementation of the same primitive.

4.1 Introduction

As indicated in Chapter 1, the possibility of implementing most cryptographic primitives in NC^0 was left wide open. We present a positive answer to this basic question, showing that surprisingly many cryptographic tasks can be performed in constant parallel time. Since the existence of cryptographic primitives implies that $\mathrm{P} \neq \mathrm{NP}$, we cannot expect unconditional results and have to rely on some unproven assumptions.[1] However, we avoid relying on *specific* intractability assumptions. Instead, we assume the existence of cryptographic primitives in a relatively "high" complexity class and transform them to the seemingly degenerate complexity class NC^0 without substantial loss of their cryptographic strength. These transformations are inherently non-black-box, thus providing further evidence for the usefulness of non-black-box techniques in cryptography.

4.1.1 Previous Work

Linial et al. show that pseudorandom *functions* cannot be computed even in AC^0 [LMN93].[2] However, no such impossibility result is known for pseudorandom generators (PRGs). The existence of PRGs in NC^0 has been recently studied in [CM01, MST03]. Cryan and Miltersen [CM01] observe that there is no PRG in NC^0_2, and prove that there is no PRG in NC^0_3 achieving a superlinear stretch; namely, one that stretches n bits to $n + \omega(n)$ bits.[3] Mossel et al. [MST03] extend this

[1]This is not the case for non-cryptographic PRGs such as ϵ-biased generators, for which we do obtain unconditional results.

[2]In fact, the results of [LMN93] show that any pseudorandom function in AC^0 can be broken in quasipolynomial time (i.e., in time $O(n^{\mathrm{polylog}(n)})$). Hence, they do not rule out the existence of very weak pseudorandom functions, e.g., ones that cannot be broken in time $O(n^{\log\log(n)})$.

[3]From here on, we use a crude classification of PRGs into ones having sublinear, linear, or superlinear additive stretch. Note that a PRG stretching its seed by just one bit can be invoked *in parallel* (on seeds of length n^ϵ) to yield a PRG stretching its seed by $n^{1-\epsilon}$ bits, for an arbitrary $\epsilon > 0$. Also, an NC^0 PRG with some linear stretch can be

impossibility to NC^0_4. Viola [Vio05] shows that a PRG in AC^0 with superlinear stretch cannot be obtained from a one-way function (OWF) via non-adaptive black-box constructions. This result can be extended to rule out such a construction even if we start with a PRG whose stretch is sublinear. Negative results for other restricted computation models appear in [GKY89, YY94].

On the positive side, Impagliazzo and Naor [IN96] construct a (sublinear-stretch) PRG in AC^0, relying on an intractability assumption related to the subset-sum problem. PRG candidates in NC^1 (or even TC^0) are more abundant, and can be based on a variety of standard cryptographic assumptions including ones related to the intractability of factoring [Kha93, NR04], discrete logarithms [BM84, Yao82, NR04] and lattice problems [Ajt96, HILL99] (see Remark 4.5.8).[4]

Unlike the case of pseudorandom generators, the question of one-way functions in NC^0 is relatively unexplored. The impossibility of OWFs in NC^0_2 follows from the easiness of 2-SAT [Gol00, CM01]. Håstad [Hås87] constructs a family of permutations in NC^0 whose inverses are P-hard to compute. Cryan and Miltersen [CM01], improving on [AAR98], present a circuit family in NC^0_3 whose range decision problem is NP-complete. This, however, gives no evidence of cryptographic strength. Since any PRG is also a OWF, all PRG candidates cited above are also OWF candidates. (In fact, the one-wayness of an NC^1 function often serves as the underlying cryptographic *assumption.*) Finally, Goldreich [Gol00] suggests a candidate OWF in NC^0, whose conjectured security does not follow from any well-known assumption.

4.1.2 Our Results

A general compiler

Our main result is that any OWF (resp., PRG) in a relatively high complexity class, containing uniform NC^1 and even $\oplus L/poly$, can be efficiently "compiled" into a corresponding OWF (resp., sublinear-stretch PRG) in NC^0_4. The existence of OWF and PRG in this class is a mild assumption, implied in particular by most number-theoretic or algebraic intractability assumptions commonly used in cryptography. Hence, the existence of OWF and sublinear-stretch PRG in NC^0 follows from a variety of standard assumptions and is not affected by the potential weakness of a particular algebraic structure. It is important to note that the PRG produced by our compiler will generally have a sublinear additive stretch even if the original PRG has a large stretch. However, one cannot do much better when insisting on an NC^0_4 PRG, as there is no PRG with superlinear stretch in NC^0_4 [MST03].

The above results extend to other cryptographic primitives including one-way permutation, encryption, signature, commitment, and collision-resistant hashing. Aiming at NC^0 implementations, we can use our machinery in two different ways: (1) compile a primitive in a relatively high complexity class (say NC^1) into its randomized encoding and show that the encoding inherits the security properties of this primitive; or (2) use known *reductions* between cryptographic primitives, together with NC^0 primitives we already constructed (e.g., OWF or PRG), to obtain new NC^0 primitives. Of course, this approach is useful only when the reduction itself is in NC^0.[5] We mainly

composed with itself a constant number of times to yield an NC^0 PRG with an arbitrary linear stretch.

[4]In some of these constructions it seems necessary to allow a *collection* of NC^1 PRGs, and use polynomial-time preprocessing to pick (once and for all) a random instance from this collection. This is similar to the more standard notion of OWF collection (cf. [Gol01a, Section 2.4.2]). See Appendix A for further discussion of this slightly relaxed notion of PRG.

[5]If the reduction is in NC^1 one can combine the two approaches: first apply the NC^1 reduction to an NC^0 primitive of type X that was already constructed (e.g., OWF or PRG) to obtain a new NC^1 primitive of type Y, and then use

adopt the first approach, since most of the known reductions between primitives are not in NC^0. (An exception in the case of symmetric encryption and zero-knowledge proofs will be discussed in Section 4.7.)

A caveat. It is important to note that in the case of two-party primitives (such as encryption schemes, signatures, commitments and zero-knowledge proofs) our compiler yields an NC^0 sender (i.e., encrypting party, committing party, signer or prover, according to the case) but does not promise anything regarding the parallel complexity of the receiver (the decrypting party or verifier). In fact, we prove that, in all these cases, it is *impossible* to implement the receiver in NC^0, regardless of the complexity of the sender. An interesting feature of the case of commitment is that we can also improve the parallel complexity at the receiver's end. Specifically, our receiver can be realized by an NC^0 circuit with a single unbounded fan-in AND gate at the top. The same holds for applications of commitment such as coin-flipping and zero-knowledge proofs.

Parallel reductions

In some cases our techniques yield NC^0 reductions between different cryptographic primitives. (Unlike the results discussed above, here we consider *unconditional* reductions that do not rely on unproven assumptions.) Specifically, we get new NC^0 constructions of PRG and non-interactive commitment from one-to-one OWF. (In fact, in the case of PRG the reduction holds even with more general types of one-way functions.) These reductions are obtained by taking a standard NC^1 reduction of a primitive $\mathcal{P}$ from a primitive $\mathcal{P}'$, and applying our compiler to it. This works only for reductions that makes non-adaptive calls to $\mathcal{P}'$. The standard reductions of PRGs and non-interactive commitments from one-to-one OWF (cf. [HILL99, Blu83, GL89]) admit such a structure, and thus can be compiled into NC^0 reductions. (We remark that if the original reduction uses the underlying primitive $\mathcal{P}'$ as a black-box than so does the new reduction. This should not be confused with the fact that the NC^0 reduction is obtained by using the "code" of the original reduction. Indeed, all the the NC^0 reductions obtained in this chapter are black-box reductions.)

Non-cryptographic generators

Our techniques can also be applied to obtain unconditional constructions of non-cryptographic PRGs. In particular, building on an ϵ-biased generator in NC^0_5 constructed by Mossel et al. [MST03], we obtain a linear-stretch ϵ-biased generator in NC^0_3. This generator has optimal locality, answering an open question posed in [MST03]. It is also essentially optimal with respect to stretch, since locality 3 does not allow for a superlinear stretch [CM01]. Our techniques apply also to other types of non-cryptographic PRGs such as generators for space-bounded computation [BNS89, Nis92], yielding such generators (with sublinear stretch) in NC^0_3.

4.1.3 Overview of Techniques

Our key observation is that instead of computing a given "cryptographic" function $f(x)$, it might suffice to compute a randomized encoding $\hat{f}(x,r)$ of f. Recall that $\hat{f}(x,r)$ has the following relation to f:

the first approach to compile the latter primitive into an NC^0 primitive (of type Y). As in the first approach, this construction requires to prove that a randomized encoding of a primitive Y preserves its security.

1. (Correctness.) For every fixed input x and a uniformly random choice of r, the output distribution $\hat{f}(x, r)$ forms a "randomized encoding" of $f(x)$, from which $f(x)$ can be decoded. That is, if $f(x) \neq f(x')$ then the random variables $\hat{f}(x, r)$ and $\hat{f}(x', r')$, induced by a uniform choice of r, r', should have disjoint supports.

2. (Privacy.) There exists a simulator algorithm that given $f(x)$ samples from the distribution $\hat{f}(x, r)$ induced by a uniform choice of r. That is, the distribution $\hat{f}(x, r)$ hides all the information about x except for the value $f(x)$. In particular, if $f(x) = f(x')$ then the random variables $\hat{f}(x, r)$ and $\hat{f}(x', r')$ should be identically distributed.

Each of these requirements alone can be satisfied by a trivial function $\hat{f}$ (e.g., $\hat{f}(x, r) = x$ and $\hat{f}(x, r) = 0$, respectively). However, the combination of the two requirements can be viewed as a non-trivial natural relaxation of the usual notion of computing. In a sense, the function $\hat{f}$ defines an "information-theoretically equivalent" representation of f.

For this approach to be useful in our context, two conditions should be met. First, we need to argue that a randomized encoding $\hat{f}$ can be *securely* used as a substitute for f. Second, we hope that this relaxation is sufficiently *liberal*, in the sense that it allows to efficiently encode relatively complex functions f by functions $\hat{f}$ in NC^0. These two issues are addressed in the following subsections.

Security of Randomized Encodings

To illustrate how a randomized encoding $\hat{f}$ can inherit the security features of f, consider the case where f is a OWF. We argue that the hardness of inverting $\hat{f}$ reduces to the hardness of inverting f. Indeed, a successful algorithm A for inverting $\hat{f}$ can be used to successfully invert f as follows: given an output y of f, apply the efficient sampling algorithm guaranteed by requirement 2 to obtain a random encoding $\hat{y}$ of y. Then, use A to obtain a preimage (x, r) of $\hat{y}$ under $\hat{f}$, and output x. It follows from requirement 1 that x is indeed a preimage of y under f. Moreover, if y is the image of a uniformly random x, then $\hat{y}$ is the image of a uniformly random pair (x, r). Hence, the success probability of inverting f is the same as that of inverting $\hat{f}$.

The above argument can tolerate some relaxations to the notion of randomized encoding. In particular, one can relax the second requirement to allow the distributions $\hat{f}(x, r)$ and $\hat{f}(x', r')$ be only *statistically close*, or even *computationally indistinguishable.* On the other hand, to maintain the security of other cryptographic primitives, it may be required to further strengthen this notion. For instance, when f is a PRG, the above requirements do not guarantee that the output of $\hat{f}$ is pseudo-random, or even that its output is longer than its input. However, by imposing suitable "regularity" requirements (e.g., the properties of *perfect* encoding cf. Definition 3.1.5) on the output encoding defined by $\hat{f}$, it can be guaranteed that if f is a PRG then so is $\hat{f}$. Thus, different security requirements suggest different variations of the above notion of randomized encoding.

Complexity of Randomized Encodings

It remains to address the second issue: can we encode a complex function f by an NC^0 function $\hat{f}$? Our best solutions to this problem rely on the machinery of *randomizing polynomials,* described below. But first, we outline a simple alternative approach[6] based on Barrington's theorem [Bar86],

[6]In fact, a modified version of this approach has been applied for constructing randomizing polynomials in [CFIK03].

combined with a randomization technique of Kilian [Kil88].

Suppose f is a boolean function in NC^1. (Non-boolean functions are handled by concatenation. See Lemma 3.2.1.) By Barrington's theorem, evaluating $f(x)$, for such a function f, reduces to computing an iterated product of polynomially many elements $s_1, \ldots, s_m$ from the symmetric group S_5, where each s_i is determined by a single bit of x (i.e., for every i there exists j such that s_i is a function of x_j). Now, let $\hat{f}(x, r) = (s_1 r_1, r_1^{-1} s_2 r_2, \ldots, r_{m-2}^{-1} s_{m-1} r_{m-1}, r_{m-1}^{-1} s_m)$, where the random inputs r_i are picked uniformly and independently from S_5. It is not hard to verify that the output $(t_1, \ldots, t_m)$ of $\hat{f}$ is random subject to the constraint that $t_1 t_2 \cdots t_m = s_1 s_2 \cdots s_m$, where the latter product is in one-to-one correspondence to $f(x)$. It follows that $\hat{f}$ is a randomized encoding of f. Moreover, $\hat{f}$ has constant locality when viewed as a function over the alphabet S_5, and thus yields the qualitative result we are after.

However, the above construction falls short of providing a randomized encoding in NC^0, since it is impossible to sample a uniform element of S_5 in NC^0 (even up to a negligible statistical distance).[7] Also, this $\hat{f}$ does not satisfy the extra "regularity" properties required by more "sensitive" primitives such as PRGs or one-way permutations. The solutions presented next avoid these disadvantages and, at the same time, apply to a higher complexity class than NC^1 and achieve a very small constant locality.

Randomizing polynomials. The concept of randomizing polynomials was introduced by Ishai and Kushilevitz [IK00] as a representation of functions by vectors of low-degree multivariate polynomials. (Interestingly, this concept was motivated by questions in the area of *information-theoretic* secure multiparty computation, which seems unrelated to the current context.) Randomizing polynomials capture the above encoding question within an algebraic framework. Specifically, a representation of $f(x)$ by randomizing polynomials is a randomized encoding $\hat{f}(x, r)$ as defined above, in which x and r are viewed as vectors over a finite field $\mathbb{F}$ and the outputs of $\hat{f}$ as multivariate polynomials in the variables x and r. In this work, we will always let $\mathbb{F} = \mathbb{F}_2$.

The most crucial parameter of a randomizing polynomials representation is its algebraic *degree*, defined as the maximal (total) degree of the outputs (i.e., the output multivariate polynomials) as a function of the input variables in x and r. (Note that both x and r count towards the degree.) Quite surprisingly, it is shown in [IK00, IK02] that every boolean function $f : \{0,1\}^n \to \{0,1\}$ admits a representation by *degree-3* randomizing polynomials whose number of inputs and outputs is at most *quadratic* in its (mod-2) branching program size (cf. Section 2.3). (Moreover, this degree bound is tight in the sense that most boolean functions do not admit a degree-2 representation.) Recall that a representation of a non-boolean function can be obtained by concatenating representations of its output bits, using independent blocks of random inputs. (See Lemma 3.2.1.) This concatenation leaves the degree unchanged.

The above positive result implies that functions whose output bits can be computed in the complexity class $\oplus\mathrm{L}/poly$ admit an efficient representation by degree-3 randomizing polynomials. This also holds if one requires the most stringent notion of representation required by our applications (i.e., perfect encoding). We note, however, that different constructions from the literature [IK00, IK02, CFIK03] are incomparable in terms of their exact efficiency and the security-preserving features they satisfy. Hence, different constructions may be suitable for different applications.

[7] Barrington's theorem generalizes to apply over arbitrary non-solvable groups. Unfortunately, there are no such groups whose order is a power of two.

Degree vs. locality. Combining our general methodology with the above results on randomizing polynomials already brings us close to our goal, as it enables "degree-3 cryptography". Taking on from here, we show that any function $f : \{0,1\}^n \to \{0,1\}^m$ of algebraic degree d admits an efficient randomized encoding $\hat{f}$ of (degree d and) locality $d+1$. That is, each output bit of $\hat{f}$ can be computed by a degree-d polynomial over $\mathbb{F}_2$ depending on at most $d+1$ inputs and random inputs. Combined with the previous results, this allows us to make the final step from degree 3 to locality 4.

4.1.4 Organization

In Section 4.2 we construct a perfect and statistical encoding in NC^0_4 for various log-space classes (i.e., $\oplus\mathrm{L}/poly$ and $\mathrm{NL}/poly$). We then apply randomized encodings to obtain NC^0 implementations of different primitives: OWFs (Section 4.4), cryptographic and non-cryptographic PRGs (Section 4.5), collision-resistent hashing (Section 4.6), public-key and symmetric encryption (Section 4.7) and other cryptographic primitives (Section 4.8). We conclude in Section 4.9 with a summary of the results obtained in this chapter.

4.2 Randomized Encoding in NC^0 for Functions in $\oplus\mathrm{L}/poly$,$\mathrm{NL}/poly$

In this section we construct randomized encodings in NC^0. We first review a construction from [IK02] of degree-3 randomizing polynomials based on mod-2 branching programs and analyze some of its properties. Next, we introduce a general locality reduction technique, allowing to transform a degree-d encoding to a $(d+1)$-local encoding. Finally, we discuss extensions to other types of BPs.

Degree-3 randomizing polynomials from mod-2 branching programs [IK02]. Let $BP = (G, \phi, s, t)$ be a mod-2 BP of size ℓ, computing a boolean[8] function $f : \{0,1\}^n \to \{0,1\}$; that is, $f(x) = 1$ if and only if the number of paths from s to t in G_x equals 1 modulo 2. Fix some topological ordering of the vertices of G, where the source vertex s is labeled 1 and the terminal vertex t is labeled ℓ. Let $A(x)$ be the $\ell \times \ell$ adjacency matrix of G_x viewed as a formal matrix whose entries are degree-1 polynomials in the input variables x. Specifically, the (i,j) entry of $A(x)$ contains the value of $\phi(i,j)$ on x if (i,j) is an edge in G, and 0 otherwise. (Hence, $A(x)$ contains the constant 0 on and below the main diagonal, and degree-1 polynomials in the input variables above the main diagonal.) Define $L(x)$ as the submatrix of $A(x) - I$ obtained by deleting column s and row t (i.e., the first column and the last row). As before, each entry of $L(x)$ is a degree-1 polynomial in a single input variable x_i; moreover, $L(x)$ contains the constant -1 in each entry of its second diagonal (the one below the main diagonal) and the constant 0 below this diagonal. (See Figure 4.2.1.)

Fact 4.2.1 ([IK02]) *$f(x) = \det(L(x))$, where the determinant is computed over $\mathbb{F}_2$.*

Proof sketch: Since G is acyclic, the number of $s-t$ paths in G_x mod 2 can be written as $(I + A(x) + A(x)^2 + \ldots + A(x)^\ell)_{s,t} = (I - A(x))^{-1}_{s,t}$ where I denotes an $\ell \times \ell$ identity matrix and all arithmetic is over $\mathbb{F}_2$. Recall that $L(x)$ is the submatrix of $A(x) - I$ obtained by deleting column s

[8]The following construction generalizes naturally to a (counting) mod-p BP, computing a function $f : \{0,1\}^n \to Z_p$. In this work, however, we will only be interested in the case $p = 2$.

$$\begin{pmatrix} 1 & r^{(1)}_1 & r^{(1)}_2 & \cdot & \cdot & r^{(1)}_{\ell-2} \\ 0 & 1 & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 1 & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & 1 & \cdot & \cdot \\ 0 & 0 & 0 & 0 & 1 & r^{(1)}_{\binom{\ell-1}{2}} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} * & * & * & * & * & * \\ -1 & * & * & * & * & * \\ 0 & -1 & * & * & * & * \\ 0 & 0 & -1 & * & * & * \\ 0 & 0 & 0 & -1 & * & * \\ 0 & 0 & 0 & 0 & -1 & * \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & r^{(2)}_1 \\ 0 & 1 & 0 & 0 & 0 & r^{(2)}_2 \\ 0 & 0 & 1 & 0 & 0 & \cdot \\ 0 & 0 & 0 & 1 & 0 & \cdot \\ 0 & 0 & 0 & 0 & 1 & r^{(2)}_{\ell-2} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 4.2.1: The matrices $R_1(r^{(1)}), L(x)$ and $R_2(r^{(2)})$ (from left to right). The symbol $*$ represents a degree-1 polynomial in an input variable.

and row t. Hence, expressing $(I - A(x))^{-1}_{s,t}$ using the corresponding cofactor of $I - A(x)$, we have:

$$\begin{aligned} (I - A(x))^{-1}_{s,t} &= (-1)^{s+t} \frac{\det(-L(x))}{\det(I - A(x))} \\ &= \det L(x). \end{aligned}$$

■

Let $r^{(1)}$ and $r^{(2)}$ be vectors over $\mathbb{F}_2$ of length $\sum_{i=1}^{\ell-2} i = \binom{\ell-1}{2}$ and $\ell - 2$, respectively. Let $R_1(r^{(1)})$ be an $(\ell-1) \times (\ell-1)$ matrix with 1's on the main diagonal, 0's below it, and $r^{(1)}$'s elements in the remaining $\binom{\ell-1}{2}$ entries above the diagonal (a unique element of $r^{(1)}$ is assigned to each matrix entry). Let $R_2(r^{(2)})$ be an $(\ell-1) \times (\ell-1)$ matrix with 1's on the main diagonal, $r^{(2)}$'s elements in the rightmost column, and 0's in each of the remaining entries. (See Figure 4.2.1.)

Fact 4.2.2 ([IK02]) *Let M, M' be $(\ell-1) \times (\ell-1)$ matrices that contain the constant -1 in each entry of their second diagonal and the constant 0 below this diagonal. Then, $\det(M) = \det(M')$ if and only if there exist $r^{(1)}$ and $r^{(2)}$ such that $R_1(r^{(1)}) M R_2(r^{(2)}) = M'$.*

Proof sketch: Suppose that $R_1(r^{(1)}) M R_2(r^{(2)}) = M'$ for some $r^{(1)}$ and $r^{(2)}$. Then, since $\det(R_1(r^{(1)})) = \det(R_2(r^{(2)})) = 1$, it follows that $\det(M) = \det(M')$.

For the second direction assume that $\det(M) = \det(M')$. We show that there there exist $r^{(1)}$ and $r^{(2)}$ such that $R_1(r^{(1)}) M R_2(r^{(2)}) = M'$. Multiplying M by a matrix $R_1(r^{(1)})$ on the left is equivalent to adding to each row of M a linear combination of the rows below it. On the other hand, multiplying M by a matrix $R_2(r^{(2)})$ on the right is equivalent to adding to the last column of M a linear combination of the other columns. Observe that a matrix M that contains the constant -1 in each entry of its second diagonal and the constant 0 below this diagonal can be transformed, using such left and right multiplications, to a canonic matrix H_y containing -1's in its second diagonal, an arbitrary value y in its top-right entry, and 0's elsewhere. Since $\det(R_1(r^{(1)})) = \det(R_2(r^{(2)})) = 1$, we have $\det(M) = \det(H_y) = y$. Thus, when $\det(M) = \det(M') = y$ we can write $H_y = R_1(r^{(1)}) M R_2(r^{(2)}) = R_1(s^{(1)}) M' R_2(s^{(2)})$ for some $r^{(1)}, r^{(2)}, s^{(1)}, s^{(2)}$. Multiplying both sides by $R_1(s^{(1)})^{-1}, R_2(s^{(2)})^{-1}$, and observing that each set of matrices $R_1(\cdot)$ and $R_2(\cdot)$ forms a multiplicative group finishes the proof. ■

Lemma 4.2.3 (implicit in [IK02]) *Let BP be a mod-2 branching program computing the boolean function f. Define a degree-3 function $\hat{f}(x,(r^{(1)},r^{(2)}))$ whose outputs contain the $\binom{\ell}{2}$ entries on or above the main diagonal of the matrix $R_1(r^{(1)})L(x)R_2(r^{(2)})$. Then, $\hat{f}$ is a perfect randomized encoding of f.*

Proof: We start by showing that the encoding is stretch preserving. The length of the random input of $\hat{f}$ is $m = \binom{\ell-1}{2} + \ell - 2 = \binom{\ell}{2} - 1$ and its output length is $s = \binom{\ell}{2}$. Thus we have $s = m+1$, and since f is a boolean function its encoding $\hat{f}$ preserves its stretch.

We now describe the decoder and the simulator. Given an output of $\hat{f}$, representing a matrix M, the decoder B simply outputs $\det(M)$. (Note that the entries below the main diagonal of this matrix are constants and therefore are not included in the output of $\hat{f}$.) By Facts 4.2.1 and 4.2.2, $\det(M) = \det(L(x)) = f(x)$, hence the decoder is perfect.

The simulator S, on input $y \in \{0,1\}$, outputs the $\binom{\ell}{2}$ entries on and above the main diagonal of the matrix $R_1(r^{(1)})H_yR_2(r^{(2)})$, where $r^{(1)}$, $r^{(2)}$ are randomly chosen, and H_y is the $(\ell-1)\times(\ell-1)$ matrix that contains -1's in its second diagonal, y in its top-right entry, and 0's elsewhere.

By Facts 4.2.1 and 4.2.2, for every $x \in \{0,1\}^n$ the supports of $\hat{f}(x,U_m)$ and of $S(f(x))$ are equal. Specifically, these supports include all strings in $\{0,1\}^s$ representing matrices with determinant $f(x)$. Since the supports of $S(0)$ and $S(1)$ form a disjoint partition of the entire space $\{0,1\}^s$ (by Fact 4.2.2) and since S uses $m = s-1$ random bits, it follows that $|\text{support}(S(b))| = 2^m$, for $b \in \{0,1\}$. Since both the simulator and the encoding use m random bits, it follows that both distributions, $\hat{f}(x,U_m)$ and $S(f(x))$, are uniform over their support and therefore are equivalent. Finally, since the supports of $S(0)$ and $S(1)$ halve the range of $\hat{f}$ (that is, $\{0,1\}^s$), the simulator is also balanced. ■

Reducing the locality. It remains to convert the degree-3 encoding into one in NC^0. To this end, we show how to construct for any degree-d function (where d is constant) a $(d+1)$-local perfect encoding. Using the composition lemma, we can obtain an NC^0 encoding of a function by first encoding it as a constant-degree function, and then applying the locality construction.

The idea for the locality construction is to represent a degree-d polynomial as a sum of monomials, each having locality d, and randomize this sum using a variant of the method for randomizing group product, described in Section 4.1.3. (A direct use of the latter method over the group Z_2 gives a $(d+2)$-local encoding instead of the $(d+1)$-local one obtained here.)

Construction 4.2.4 (Locality construction) *Let $f(x) = T_1(x)+\ldots+T_k(x)$, where $f, T_1, \ldots, T_k : \mathbb{F}_2^n \to \mathbb{F}_2$ and summation is over $\mathbb{F}_2$. The local encoding $\hat{f} : \mathbb{F}_2^{n+(2k-1)} \to \mathbb{F}_2^{2k}$ is defined by:*

$$\begin{aligned}\hat{f}(x,(r_1,\ldots,r_k,r'_1,\ldots,r'_{k-1})) \overset{def}{=} (&T_1(x)-r_1, T_2(x)-r_2, \ldots, T_k(x)-r_k,\\ &r_1-r'_1, r'_1+r_2-r'_2, \ldots, r'_{k-2}+r_{k-1}-r'_{k-1}, r'_{k-1}+r_k).\end{aligned}$$

For example, applying the locality construction to the polynomial $x_1x_2 + x_2x_3 + x_4$ results in the encoding $(x_1x_2 - r_1, x_2x_3 - r_2, x_4 - r_3, r_1 - r'_1, r'_1 + r_2 - r'_2, r'_2 + r_3)$.

Lemma 4.2.5 (Locality lemma) *Let f and $\hat{f}$ be as in Construction 4.2.4. Then, $\hat{f}$ is a perfect randomized encoding of f. In particular, if f is a degree-d polynomial written as a sum of monomials, then $\hat{f}$ is a perfect encoding of f with degree d and locality* $\max(d+1,3)$.

Proof: Since $m = 2k-1$ and $s = 2k$, the encoding $\hat{f}$ is stretch preserving. Moreover, given $\hat{y} = \hat{f}(x, r)$ we can decode the value of $f(x)$ by summing up the bits of $\hat{y}$. It is not hard to verify that such a decoder never errs. To prove perfect privacy we define a simulator as follows. Given $y \in \{0, 1\}$, the simulator S uniformly chooses $2k-1$ random bits $r_1, \ldots, r_{2k-1}$ and outputs $(r_1, \ldots, r_{2k-1}, y - (r_1 + \ldots + r_{2k-1}))$. Obviously, $S(y)$ is uniformly distributed over the $2k$-length strings whose bits sum up to y over $\mathbb{F}_2$. It thus suffices to show that the outputs of $\hat{f}(x, U_m)$ are uniformly distributed subject to the constraint that they add up to $f(x)$. This follows by observing that, for any x and any assignment $w \in \{0, 1\}^{2k-1}$ to the first $2k-1$ outputs of $\hat{f}(x, U_m)$, there is a unique way to set the random inputs r_i, r'_i so that the output of $\hat{f}(x, (r, r'))$ is consistent with w. Indeed, for $1 \leq i \leq k$, the values of x, w_i uniquely determine r_i. For $1 \leq i \leq k-1$, the values w_{k+i}, r_i, r'_{i-1} determine r'_i (where $r'_0 \overset{\text{def}}{=} 0$). Therefore, $S(f(x)) \equiv \hat{f}(x, U_m)$. Moreover, S is balanced since the supports of $S(0)$ and $S(1)$ halve $\{0, 1\}^s$ and $S(y)$ is uniformly distributed over its support for $y \in \{0, 1\}$. ■

In Section 4.3 we describe a graph-based generalization of Construction 4.2.4, which in some cases can give rise to a (slightly) more compact encoding $\hat{f}$.

We now present the main theorem of this section.

Theorem 4.2.6 $\oplus\mathrm{L}/poly \subseteq \mathcal{PREN}$. *Moreover, any* $f \in \oplus\mathrm{L}/poly$ *admits a perfect randomized encoding in* NC^0_4 *whose degree is 3.*

Proof: The theorem is derived by combining the degree-3 construction of Lemma 4.2.3 together with the Locality Lemma (4.2.5), using the Composition Lemma (3.2.3) and the Concatenation Lemma (3.2.2). ■

Remark 4.2.7 An alternative construction of perfect randomized encodings in NC^0 can be obtained using a randomizing polynomials construction from [IK02, Sec. 3], which is based on an information-theoretic variant of Yao's garbled circuit technique [Yao86]. This construction yields an encoding with a (large) constant locality, without requiring an additional "locality reduction" step (of Construction 4.2.4). This construction is weaker than the current one in that it only efficiently applies to functions in NC^1 rather than $\oplus\mathrm{L}/poly$. For functions in NC^1, the complexity of this alternative (in terms of randomness and output length) is incomparable to the complexity of the current construction.

There are variants of the above construction that can handle non-deterministic branching programs as well, at the expense of losing perfectness [IK00, IK02]. In particular, Theorem 2 in [IK02] encodes non-deterministic branching programs by perfectly-correct statistically-private functions of degree 3 (over $\mathbb{F}_2$). Hence, by using Lemmas 4.2.5, 3.2.3, we get a perfectly-correct statistically-private encoding in NC^0_4 for functions in $\mathrm{NL}/poly$. (In fact, we can also use [IK00, IK02] to obtain *perfectly-private* statistically-correct encoding in NC^0_4 for non-deterministic branching programs.) Based on the above, we get the following theorem:

Theorem 4.2.8 $\mathrm{NL}/poly \subseteq \mathcal{SREN}$. *Moreover, any* $f \in \mathrm{NL}/poly$ *admits a perfectly-correct statistically-private randomized encoding in* NC^0_4.

We can rely on Theorem 4.2.6 to derive the following corollary.

Corollary 4.2.9 *Any function f in $\mathcal{PREN}$ (resp., $\mathcal{SREN}$, $\mathcal{CREN}$) admits a perfect (resp., statistical, computational) randomized encoding of degree 3 and locality 4 (i.e., in NC^0_4).*

Proof: We first encode f by a perfect (resp., statistical, computational) encoding $\hat{f}$ in NC^0, guaranteed by the fact that f is in $\mathcal{PREN}$ (resp., $\mathcal{SREN}$, $\mathcal{CREN}$). Then, since $\hat{f}$ is in $\oplus\mathrm{L}/poly$, we can use Theorem 4.2.6 to perfectly encode $\hat{f}$ by a function $\hat{f}'$ in NC^0_4 whose degree is 3. By the Composition Lemmas (3.2.3, 3.2.4), $\hat{f}'$ perfectly (resp. statistically, computationally) encodes the function f. ■

4.3 A Generalization of the Locality Construction

In the Locality Construction (4.2.4), we showed how to encode a degree d function by an NC^0_{d+1} encoding. We now describe a graph based construction that generalizes the previous one. The basic idea is to view the encoding $\hat{f}$ as a graph. The nodes of the graph are labeled by terms of f and the edges by random inputs of $\hat{f}$. With each node we associate an output of $\hat{f}$ in which we add to its term the labels of the edges incident to the node. Formally,

Construction 4.3.1 (General locality construction) *Let $f(x) = T_1(x) + \ldots + T_k(x)$, where $f, T_1, \ldots, T_k : \mathbb{F}_2^n \to \mathbb{F}_2$ and summation is over $\mathbb{F}_2$. Let $G = (V, E)$ be a directed graph with k nodes $V = \{1, \ldots, k\}$ and m edges. The encoding $\hat{f}_G : \mathbb{F}_2^{n+m} \to \mathbb{F}_2^k$ is defined by:*

$$\hat{f}_G(x, (r_{i,j})_{(i,j)\in E}) \overset{def}{=} \left(T_i(x) + \sum_{j|(j,i)\in E} r_{j,i} - \sum_{j|(i,j)\in E} r_{i,j} \right)_{i=1}^{k}.$$

From here on, we will identify with the directed graph G its underlying undirected graph. The above construction yields a perfect encoding when G is a tree (see Lemma 4.3.2 below). The locality of an output bit of $\hat{f}_G$ is the locality of the corresponding term plus the degree of the node in the graph. The locality construction described in Construction 4.2.4 attempts to minimize the maximal locality of a node in the graph; hence it adds k "dummy" 0 terms to f and obtains a tree in which all of the k non-dummy terms of f are leaves, and the degree of each dummy term is at most 3. When the terms of f vary in their locality, a more compact encoding $\hat{f}$ can be obtained by increasing the degree of nodes which represent terms with lower locality.

Lemma 4.3.2 (Generalized locality lemma) *Let f and $\hat{f}_G$ be as in Construction 4.3.1. Then,*

1. *$\hat{f}_G$ is a perfectly correct encoding of f.*
2. *If G is connected, then $\hat{f}_G$ is also a balanced encoding of f (and in particular it is perfectly private).*
3. *If G is a tree, then $\hat{f}_G$ is also stretch preserving; that is, $\hat{f}_G$ perfectly encodes f.*

Proof: (1) Given $\hat{y} = \hat{f}_G(x, r)$ we decode $f(x)$ by summing up the bits of $\hat{y}$. Since each random variable $r_{i,j}$ appears only in the i-th and j-th output bits, it contributes 0 to the overall sum and therefore the bits of $\hat{y}$ always add up to $f(x)$.

To prove (2) we use the same simulator as in the locality construction (see proof of Lemma 4.2.5). Namely, given $y \in \{0,1\}$, the simulator S chooses $k-1$ random bits $r_1, \ldots, r_{k-1}$ and outputs $(r_1, \ldots, r_{k-1}, y - (r_1 + \ldots + r_{k-1}))$. This simulator is balanced since the supports of $S(0)$ and $S(1)$ halve $\{0,1\}^k$ and $S(y)$ is uniformly distributed over its support for $y \in \{0,1\}$. We now prove that $\hat{f}_G(x, U_m) \equiv S(f(x))$. Since the support of $S(f(x))$ contains exactly 2^{k-1} strings (namely, all k-bit strings whose bits sum up to $f(x)$), it suffices to show that for any input x and output $w \in \text{support}(S(f(x)))$ there are $2^m/2^{k-1}$ random inputs r such that $\hat{f}_G(x,r) = w$. (Note that $m \geq k-1$ since G is connected.) Let $T \subseteq E$ be a spanning tree of G. We argue that for any assignment to the $m-(k-1)$ random variables that correspond to edges in $E \setminus T$ there exists an assignment to the other random variables that is consistent with w and x. Fix some assignment to the edges in $E \setminus T$. We now recursively assign values to the remaining edges. In each step we make sure that some leaf is consistent with w by assigning the corresponding value to the edge connecting this leaf to the graph. Then, we prune this leaf and repeat the above procedure. Formally, let i be a leaf which is connected to T by an edge $e \in T$. Assume, without loss of generality, that e is an incoming edge for i. We set r_e to $w_i - (T_i(x) + \sum_{j|(j,i) \in E\setminus T} r_{j,i} - \sum_{j|(i,j)\in E \setminus T} r_{i,j})$, and remove i from T. By this we ensure that the i-th bit of $\hat{f}_G(x,r)$ is equal to w_i. (This equality will not be violated by the following steps as i is removed from T.) We continue with the above step until the tree consists of one node. Since the outputs of $\hat{f}_G(x,r)$ always sum up to $f(x)$ it follows that this last bit of $\hat{f}_G(x,r)$ is equal to the corresponding bit of w. Thus, there are at least $2^{|E\setminus T|} = 2^{m-(k-1)}$ values of r that lead to w as required.

Finally, to prove (3) note that when G is a tree we have $m = k-1$, and therefore the encoding is stretch preserving; combined with (1) and (2) $\hat{f}_G$ is also perfect. ■

4.4 One-Way Functions in NC^0

A *one-way function* (OWF) $f : \{0,1\}^* \to \{0,1\}^*$ is a polynomial-time computable function that is hard to invert; namely, every (non-uniform) polynomial time algorithm that tries to invert f on input $f(x)$, where x is picked from U_n, succeeds only with a negligible probability. Formally,

Definition 4.4.1 (One-way function) *A function $f : \{0,1\}^* \to \{0,1\}^*$ is called a* one-way function *(OWF) if it satisfies the following two properties:*

- **Easy to compute***: There exists a deterministic polynomial-time algorithm computing $f(x)$.*
- **Hard to invert***: For every non-uniform polynomial-time algorithm, A, we have*

$$\Pr_{x \leftarrow U_n}[A(1^n, f(x)) \in f^{-1}(f(x))] \leq \text{neg}(n).$$

The function f is called weakly one-way *if the second requirement is replaced with the following (weaker) one:*

- **Slightly hard to invert***: There exists a polynomial $p(\cdot)$, such that for every (non-uniform) polynomial-time algorithm, A, and all sufficiently large n's*

$$\Pr_{x \leftarrow U_n}[A(1^n, f(x)) \notin f^{-1}(f(x))] > \frac{1}{p(n)}.$$

The above definition naturally extends to functions whose domain is restricted to some infinite subset $I \subset \mathbb{N}$ of the possible input lengths, such as ones defined by a randomized encoding $\hat{f}$. As argued in Remark 3.1.8, such a partially defined OWF can be augmented into a fully defined OWF provided that the set I is polynomially-dense and efficiently recognizable (which is a feature of functions $\hat{f}$ obtained via a uniform encodings).

4.4.1 Key Lemmas

In the following we show that a perfectly correct and statistically (or even computationally) private randomized encoding $\hat{f}$ of a OWF f is also a OWF. The idea, as described in Section 4.1.3, is to argue that the hardness of inverting $\hat{f}$ reduces to the hardness of inverting f. The case of a randomized encoding that does not enjoy perfect correctness is more involved and will be dealt with later in this section.

Lemma 4.4.2 *Suppose that $f : \{0,1\}^* \to \{0,1\}^*$ is a one-way function and $\hat{f}(x,r)$ is a perfectly correct, computationally private encoding of f. Then $\hat{f}$, viewed as a single-argument function, is also one-way.*

Proof: Let $s = s(n), m = m(n)$ be the lengths of the output and of the random input of $\hat{f}$ respectively. Note that $\hat{f}$ is defined on input lengths of the form $n + m(n)$; we prove that it is hard to invert on these inputs. Assume, towards a contradiction, that there is an efficient adversary $\hat{A}$ inverting $\hat{f}(x,r)$ with success probability $\phi(n+m) > \frac{1}{q(n+m)}$ for some polynomial $q(\cdot)$ and infinitely many n's. We use $\hat{A}$ to construct an efficient adversary A that inverts f with similar success. On input $(1^n, y)$, the adversary A runs S, the statistical simulator of $\hat{f}$, on the input $(1^n, y)$ and gets a string $\hat{y}$ as the output of S. Next, A runs the inverter $\hat{A}$ on the input $(1^{n+m}, \hat{y})$, getting (x', r') as the output of $\hat{A}$ (i.e., $\hat{A}$ "claims" that $\hat{f}(x', r') = \hat{y}$). A terminates with output x'.

Complexity: Since S and $\hat{A}$ are both polynomial-time algorithms, and since $m(n)$ is polynomially bounded, it follows that A is also a polynomial-time algorithm.

Correctness: We analyze the success probability of A on input $(1^n, f(x))$ where $x \leftarrow U_n$. Let us assume for a moment that the simulator S is perfect. Observe that, by perfect correctness, if $f(x) \neq f(x')$ then the support sets of $\hat{f}(x, U_m)$ and $\hat{f}(x', U_m)$ are disjoint. Moreover, by perfect privacy the string $\hat{y}$, generated by $\hat{A}$, is always in the support of $\hat{f}(x, U_m)$. Hence, if $\hat{A}$ succeeds (that is, indeed $\hat{y} = \hat{f}(x', r')$) then so does A (namely, $f(x') = y$). Finally, observe that (by Fact 2.2.5) the input $\hat{y}$ on which A invokes $\hat{A}$ is distributed identically to $\hat{f}_n(U_n, U_{m(n)})$, and therefore A succeeds with probability $\geq \phi(n + m)$. Formally, we can write,

$$\begin{aligned}
\Pr_{x \leftarrow U_n}[A(1^n, f(x)) \in f^{-1}(f(x))] &\geq \Pr_{x \leftarrow U_n, \hat{y} \leftarrow S(1^n, f(x))}[\hat{A}(1^{n+m}, \hat{y}) \in \hat{f}^{-1}(\hat{y})] \\
&= \Pr_{x \leftarrow U_n, r \leftarrow U_{m(n)}}[\hat{A}(1^{n+m}, \hat{f}_n(x,r)) \in \hat{f}^{-1}(\hat{f}(x,r))] \\
&\geq \phi(n+m).
\end{aligned}$$

We now show that when S is computationally private, we lose only negligible success probability in the above; that is, A succeeds with probability $\geq \phi(n + m) - \text{neg}(n)$. To see this, it will be convenient to define a distinguisher D that on input $(1^n, y, \hat{y})$ computes $(x', r') = \hat{A}(1^{n+m}, \hat{y})$, and outputs 1 if $f(x') = y$ and 0 otherwise. Clearly, the success probability of A on $f(U_n)$

can be written as $\Pr_{x \leftarrow U_n}[D(1^n, f(x), S(f(x))) = 1]$. On the other hand, we showed above that $\Pr_{x \leftarrow U_n, r \leftarrow U_{m(n)}}[D(1^n, f(x), \hat{f}(x,r)) = 1] \geq \phi(n+m)$. Also, by the computational privacy of S, and Facts 2.2.10, 2.2.8, we have

$$(f(U_n), S(f(U_n))) \stackrel{c}{\equiv} (f(U_n), \hat{f}(U_n, U_{m(n)})).$$

Hence, since D is polynomial-time computable, we have

$$\begin{aligned} \Pr_{x \leftarrow U_n}[D(1^n, f(x), S(f(x))) = 1] &\geq \Pr_{x \leftarrow U_n, r \leftarrow U_{m(n)}}[D(1^n, f(x), \hat{f}(x,r)) = 1] - \mathrm{neg}(n) \\ &\geq \phi(n+m) - \mathrm{neg}(n) > \frac{1}{q(n+m)} - \mathrm{neg}(n) > \frac{1}{q'(n)}, \end{aligned}$$

for some polynomial $q'(\cdot)$ and infinitely many n's. It follows that f is not hard to invert, in contradiction to the hypothesis. ■

The efficiency of the simulator S is essential for Lemma 4.4.2 to hold. Indeed, without this requirement one could encode any one-way permutation f by the identity function $\hat{f}(x) = x$, which is obviously not one-way. (Note that the output of $\hat{f}(x)$ can be simulated inefficiently based on $f(x)$ by inverting f.)

The perfect correctness requirement is also essential for Lemma 4.4.2 to hold. To see this, consider the following example. Suppose f is a one-way permutation. Consider the encoding $\hat{f}(x,r)$ which equals $f(x)$ except if r is the all-zero string, in which case $\hat{f}(x,r) = x$. This is a statistically-correct and statistically-private encoding, but $\hat{f}$ is easily invertible since on value $\hat{y}$ the inverter can always return $\hat{y}$ itself as a possible pre-image. Still, we show below that such an $\hat{f}$ (which is only statistically correct) is a *distributionally* one-way function. We will later show how to turn a distributionally one-way function in NC^0 into a OWF in NC^0.

Definition 4.4.3 (Distributionally one-way function [IL89]) *A polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ is called* distributionally one-way *if there exists a positive polynomial $p(\cdot)$ such that for every (non-uniform) polynomial-time algorithm, A, and all sufficiently large n's,* $\mathrm{SD}((A(1^n, f(U_n)), f(U_n)), (U_n, f(U_n))) > \frac{1}{p(n)}$.

Before proving that a randomized encoding of a OWF is distributionally one-way, we need the following lemma.

Lemma 4.4.4 *Let $f, g : \{0,1\}^* \to \{0,1\}^*$ be two functions that differ on a negligible fraction of their domain; that is, $\Pr_{x \leftarrow U_n}[f(x) \neq g(x)]$ is negligible in n. Suppose that g is slightly hard to invert (but is not necessarily computable in polynomial time) and that f is computable in polynomial time. Then, f is distributionally one-way.*

Proof: Let f_n and g_n be the restrictions of f and g to n-bit inputs, that is $f = \{f_n\}, g = \{g_n\}$, and define $\varepsilon(n) \stackrel{\mathrm{def}}{=} \Pr_{x \leftarrow U_n}[f(x) \neq g(x)]$. Let $p(n)$ be the polynomial guaranteed by the assumption that g is slightly hard to invert. Assume, towards a contradiction, that f is not distributionally one-way. Then, there exists a polynomial-time algorithm, A, such that for infinitely many n's, $\mathrm{SD}((A(1^n, f_n(U_n)), f_n(U_n)), (U_n, f_n(U_n))) \leq \frac{1}{2p(n)}$. Since $(U_n, f_n(U_n)) \equiv (x', f_n(U_n))$ where $x' \leftarrow$

$f_n^{-1}(f_n(U_n))$, we get that for infinitely many n's $\mathrm{SD}((A(1^n, f_n(U_n)), f_n(U_n)), (x', f_n(U_n))) \leq \frac{1}{2p(n)}$. It follows that for infinitely many n's

$$\Pr[A(1^n, f(U_n)) \in g_n^{-1}(f_n(U_n))] \geq \Pr_{x' \leftarrow f_n^{-1}(f_n(U_n))}[x' \in g_n^{-1}(f_n(U_n))] - \frac{1}{2p(n)}. \tag{4.4.1}$$

We show that A inverts g with probability greater than $1 - \frac{1}{p(n)}$ and derive a contradiction. Specifically, for infinitely many n's we have:

$$\begin{aligned}
\Pr[B(1^n, g_n(U_n)) \in g_n^{-1}(g_n(U_n))] &\geq \Pr[B(1^n, f_n(U_n)) \in g_n^{-1}(f_n(U_n))] - \varepsilon(n) \\
&\geq \Pr_{x' \leftarrow f_n^{-1}(f_n(U_n))}[x' \in g_n^{-1}(f(U_n))] - \frac{1}{2p(n)} - \varepsilon(n) \\
&= \Pr_{x' \leftarrow f_n^{-1}(f_n(U_n))}[g_n(x') = f_n(U_n)] - \frac{1}{2p(n)} - \varepsilon(n) \\
&= \Pr_{x' \leftarrow f_n^{-1}(f_n(U_n))}[g_n(x') = f_n(x')] - \frac{1}{2p(n)} - \varepsilon(n) \\
&= 1 - \varepsilon(n) - \frac{1}{2p(n)} - \varepsilon(n) \\
&\geq 1 - \frac{1}{p(n)},
\end{aligned}$$

where the first inequality is due to the fact that f and g are ε-close, the second inequality uses (4.4.1), the second equality follows since $f(U_n) = f(x')$, the third equality is due to $x' \equiv U_n$, and the last inequality follows since ε is negligible. ■

We now use Lemma 4.4.4 to prove the distributional one-wayness of a statistically-correct encoding $\hat{f}$ based on the one-wayness of a related, perfectly correct, encoding g.

Lemma 4.4.5 *Suppose that $f : \{0,1\}^* \to \{0,1\}^*$ is a one-way function and $\hat{f}(x,r)$ is a computational randomized encoding of f. Then $\hat{f}$, viewed as a single-argument function, is distributionally one-way.*

Proof: Let B and S be the decoder and the simulator of $\hat{f}$. Define the function $\hat{g}(x,r)$ in the following way: if $B(\hat{f}(x,r)) \neq f(x)$ then $\hat{g}(x,r) = \hat{f}(x,r')$ for some r' such that $B(\hat{f}(x,r')) = f(x)$ (such an r' exists by the statistical correctness); otherwise, $\hat{g}(x,r) = \hat{f}(x,r)$. Obviously, $\hat{g}$ is a perfectly correct encoding of f (as B perfectly decodes $f(x)$ from $\hat{g}(x,r)$). Moreover, by the statistical correctness of B, we have that $\hat{f}(x,\cdot)$ and $\hat{g}(x,\cdot)$ differ only on a negligible fraction of the r's. It follows that $\hat{g}$ is also a computationally-private encoding of f (because $\hat{g}(x,U_m) \overset{s}{\equiv} \hat{f}(x,U_m) \overset{c}{\equiv} S(f(x))$). Since f is hard to invert, it follows from Lemma 4.4.2 that $\hat{g}$ is also hard to invert. (Note that $\hat{g}$ might not be computable in polynomial time; however the proof of Lemma 4.4.2 only requires that the simulator's running time and the randomness complexity of $\hat{g}$ be polynomially bounded.) Finally, it follows from Lemma 4.4.4 that $\hat{f}$ is distributionally one-way as required. ■

4.4.2 Main Results

Based on the above, we derive the main theorem of this section:

Theorem 4.4.6 *If there exists a OWF in $\mathcal{CREN}$ then there exists a OWF in* NC^0_4.

Proof: Let f be a OWF in $\mathcal{CREN}$. By Lemma 4.4.5, we can construct a distributional OWF $\hat{f}$ in NC^0, and then apply a standard transformation (cf. [IL89, Lemma 1], [Gol01a, p. 96], [Yao82]) to convert $\hat{f}$ to a OWF $\hat{f}'$ in NC^1. This transformation consists of two steps: Impagliazzo and Luby's NC^1 construction of weak OWF from distributional OWF [IL89], and Yao's NC^0 construction of a (standard) OWF from a weak OWF [Yao82] (see [Gol01a, Section 2.3]).[9] Since $\mathrm{NC}^1 \subseteq \mathcal{PREN}$ (Theorem 4.2.6), we can use Lemma 4.4.2 to encode $\hat{f}'$ by a OWF in NC^0, in particular, by one with locality 4. ■

Combining Lemmas 4.4.2, 3.2.5 and Corollary 4.2.9, we get a similar result for one-way permutations (OWPs).

Theorem 4.4.7 *If there exists a one-way permutation in $\mathcal{PREN}$ then there exists a one-way permutation in* NC^0_4.

In particular, using Theorems 4.2.6 and 4.2.8, we conclude that a OWF (resp., OWP) in NL/*poly* or $\oplus$L/*poly* (resp., $\oplus$L/*poly*) implies a OWF (resp., OWP) in NC^0_4.

Theorem 4.4.7 can be extended to trapdoor permutations (TDPs) provided that the perfect encoding satisfies the following *randomness reconstruction* property: given x and $\hat{f}(x,r)$, the randomness r can be efficiently recovered. If this is the case, then the trapdoor of f can be used to invert $\hat{f}(x,r)$ in polynomial time (but not in NC^0). Firstly, we compute $f(x)$ from $\hat{f}(x,r)$ using the decoder; secondly, we use the trapdoor-inverter to compute x from $f(x)$; and finally, we use the randomness reconstruction algorithm to compute r from x and $\hat{f}(x,r)$. The randomness reconstruction property is satisfied by the randomized encodings described in Section 4.2 and is preserved under composition and concatenation. Thus, the existence of trapdoor permutations computable in NC^0_4 follows from their existence in $\oplus$L/*poly*.

More formally, a collection of permutations $\mathcal{F} = \{f_z : D_z \to D_z\}_{z \in Z}$ is referred to as a trapdoor permutation if there exist probabilistic polynomial-time algorithms (I, D, F, F^{-1}) with the following properties. Algorithm I is an index selector algorithm that on input 1^n selects an index z from Z and a corresponding trapdoor for f_z; algorithm D is a domain sampler that on input z samples an element from the domain D_z; F is a function evaluator that given an index z and x returns $f_z(x)$; and F^{-1} is a trapdoor-inverter that given an index z, a corresponding trapdoor t and $y \in D_z$ returns $f_z^{-1}(y)$. Additionally, the collection should be hard to invert, similarly to a standard collection of one-way permutations. (For formal definition see [Gol01a, Definition 2.4.4].) By the above argument we derive the following theorem.

Theorem 4.4.8 *If there exists a trapdoor permutation $\mathcal{F}$ whose function evaluator F is in* $\oplus$L/*poly then there exists a trapdoor permutation $\hat{\mathcal{F}}$ whose function evaluator $\hat{F}$ is in* NC^0_4.

[9] We will later show a degree preserving transformation from a distributional OWF to a OWF (Lemma 8.2.3); however, in the current context the standard transformation suffices.

Remarks on Theorems 4.4.6, 4.4.7 and 4.4.8.

1. (Constructiveness) In Section 4.2, we give a constructive way of transforming a branching program representation of a function f into an NC^0 circuit computing its encoding $\hat{f}$. It follows that Theorems 4.4.6, 4.4.7 can be made constructive in the following sense: there exists a polynomial-time *compiler* transforming a branching program representation of a OWF (resp., OWP) f into an NC^0 representation of a corresponding OWF (resp., OWP) $\hat{f}$. A similar result holds for other cryptographic primitives considered in this paper.

2. (Preservation of security, a finer look) Loosely speaking, the main security loss in the reduction follows from the expansion of the input. (The simulator's running time only has a minor effect on the security, since it is added to the overall running-time of the adversary.) Thus, to achieve a level of security similar to that achieved by applying f on n-bit inputs, one would need to apply $\hat{f}$ on $n + m(n)$ bits (the random input part of the encoding does not contribute to the security). Going through our constructions (bit-by-bit encoding of the output based on some size-$\ell(n)$ BPs, followed by the locality construction), we get $m(n) = l(n) \cdot \ell(n)^{O(1)}$, where $l(n)$ is the output length of f. If the degree of all nodes in the BPs is bounded by a constant, the complexity is $m(n) = O(l(n) \cdot \ell(n)^2)$. It is possible to further reduce the overhead of randomized encoding for specific representation models, such as balanced formulas, using constructions of randomizing polynomials from [IK02, CFIK03].

3. (Generalizations) The proofs of the above theorems carry over to OWF whose security holds against efficient *uniform* adversaries (inverters). The same is true for all cryptographic primitives considered in this work. The proofs also naturally extend to the case of *collections* of OWF and OWP (see Appendix A for discussion).

4. (Concrete assumptions) The existence of a OWF in $\mathcal{SREN}$ (in fact, even in NC^1) follows from the intractability of factoring and lattice problems [Ajt96]. The existence of a OWF *collection* in $\mathcal{SREN}$ follows from the intractability of the discrete logarithm problem. Thus, we get OWFs in NC^0_4 under most standard cryptographic assumptions. In the case of OWP, we can get a collection of OWPs in NC^0_4 based on discrete logarithm [BM84, Yao82] (see also Appendix A) or RSA with a small exponent [RSA78].[10] The latter assumption is also sufficient for the construction of TDP in NC^0_4.

4.5 Pseudorandom Generators in NC^0

A *pseudorandom generator* is an efficiently computable function $G : \{0,1\}^n \rightarrow \{0,1\}^{l(n)}$ such that: (1) G has a positive stretch, namely $l(n) > n$, where we refer to the function $l(n) - n$ as the *stretch* of the generator; and (2) any "computationally restricted procedure" D, called a *distinguisher*, has a negligible advantage in distinguishing $G(U_n)$ from $U_{l(n)}$. That is, $|\Pr[D(1^n, G(U_n)) = 1] - \Pr[D(1^n, U_{l(n)}) = 1]|$ is negligible in n.

Different notions of PRGs differ mainly in the computational bound imposed on D. In the default case of *cryptographic* PRGs, D can be any non-uniform polynomial-time algorithm (alternatively, polynomial-time algorithm). In the case of ϵ-*biased* generators, D can only compute a

[10]Rabin's factoring-based OWP collection [Rab79] seems insufficient for our purposes, as it cannot be defined over the set of *all* strings of a given length. The standard modification (cf. [Gol04, p. 767]) does not seem to be in $\oplus\mathrm{L}/poly$.

linear function of the output bits, namely the exclusive-or of some subset of the bits. Other types of PRGs, e.g. for space-bounded computation, have also been considered. The reader is referred to [Gol98, Chapter 3] for a comprehensive and unified treatment of pseudorandomness.

We start by considering cryptographic PRGs. We show that a *perfect* randomized encoding of such a PRG is also a PRG. We then obtain a similar result for other types of PRGs.

4.5.1 Cryptographic Generators

Definition 4.5.1 (Pseudorandom generator) *A pseudorandom generator (PRG) is a polynomial-time computable function, $G : \{0,1\}^n \to \{0,1\}^{l(n)}$, satisfying the following two conditions:*

- **Expansion***: $l(n) > n$, for all $n \in \mathbb{N}$.*
- **Pseudorandomness***: The ensembles $\{G(U_n)\}_{n\in\mathbb{N}}$ and $\{U_{l(n)}\}_{n\in\mathbb{N}}$ are computationally indistinguishable.*

Remark 4.5.2 (PRGs with sublinear stretch) An NC^0 pseudorandom generator, G, that stretches its input by a single bit can be transformed into another NC^0 PRG, G', with stretch $l'(n)-n = n^c$ for an arbitrary constant $c < 1$. This can be done by applying G on n^c blocks of n^{1-c} bits and concatenating the results. Since the output of any PRG is computationally-indistinguishable from the uniform distribution even by a polynomial number of samples (see [Gol01a, Theorem 3.2.6]), the block generator G' is also a PRG. This PRG gains a pseudorandom bit from every block, and therefore stretches $n^c n^{1-c} = n$ input bits to $n + n^c$ output bits. Obviously, G' has the same locality as G.

Remark 4.5.3 (PRGs with linear stretch) We can also transform an NC^0_d pseudorandom generator, $G : \{0,1\}^n \to \{0,1\}^{cn}$, with some linear stretch factor $c > 1$ into another NC^0 PRG, $G'\{0,1\}^n \to \{0,1\}^{c'n}$, with arbitrary linear stretch factor $c' > 1$ and larger (but constant) output locality d'. This can be done by composing G with itself a constant number of times. That is, we let $G'(x) \stackrel{\text{def}}{=} G^{\lceil \log_c c' \rceil}(x)$ where $G^{i+1}(x) \stackrel{\text{def}}{=} G(G^i(x))$ and $G^0(x) \stackrel{\text{def}}{=} x$. Since the output of a PRG is pseudorandom even if it is invoked on a pseudorandom seed (see [Gol01a, p. 176]), the composed generator G' is also a PRG. Clearly, this PRG stretches n input bits to $c'n$ output bits and its locality is $d^{\lceil \log_c c' \rceil} = O(1)$.[11]

Remark 4.5.2 also applies to other types of generators considered in this section, and therefore we only use a crude classification of the stretch as being “sublinear”, “linear” or “superlinear”.

Lemma 4.5.4 *Suppose $G : \{0,1\}^n \to \{0,1\}^{l(n)}$ is a PRG and $\hat{G} : \{0,1\}^n \times \{0,1\}^{m(n)} \to \{0,1\}^{s(n)}$ is a perfect randomized encoding of G. Then $\hat{G}$, viewed as a single-argument function, is also a PRG.*

Proof: Since $\hat{G}$ is stretch preserving, it is guaranteed to expand its seed. To prove the pseudorandomness of its output, we again use a reducibility argument. Assume, towards a contradiction, that there exists an efficient distinguisher $\hat{D}$ that distinguishes between U_s and $\hat{G}(U_n, U_m)$ with some non-negligible advantage ϕ; i.e., ϕ such that $\phi(n+m) > \frac{1}{q(n+m)}$ for some polynomial $q(\cdot)$ and

[11]We can also increase the stretch factor by using the standard construction of Goldreich-Micali [Gol01a, Sec. 3.3.2]. In this case the locality of G' will be $d^{\lceil (c'-1)/(c-1) \rceil}$.

infinitely many n's. We use $\hat{D}$ to obtain a distinguisher D between U_l and $G(U_n)$ as follows. On input $y \in \{0,1\}^l$, run the balanced simulator of $\hat{G}$ on y, and invoke $\hat{D}$ on the resulting $\hat{y}$. If y is taken from U_l then the simulator, being balanced, outputs $\hat{y}$ that is distributed as U_s. On the other hand, if y is taken from $G(U_n)$ then, by Fact 2.2.5, the output of the simulator is distributed as $\hat{G}(U_n, U_m)$. Thus, the distinguisher D we get for G has the same advantage as the distinguisher $\hat{D}$ for $\hat{G}$. That is, the advantage of D is $\phi'(n) = \phi(n+m)$. Since $m(n)$ is polynomial, this advantage ϕ' is not only non-negligible in $n+m$ but also in n, in contradiction to the hypothesis. ■

Remark 4.5.5 (The role of balance and stretch preservation) Dropping either the balance or stretch preservation requirements, Lemma 4.5.4 would no longer hold. To see this consider the following two examples. Let G be a PRG, and let $\hat{G}(x,r) = G(x)$. Then, $\hat{G}$ is a perfectly correct, perfectly private, and balanced randomized encoding of G (the balanced simulator is $S(y) = y$). However, when r is sufficiently long, $\hat{G}$ does not expand its seed. On the other hand, we can define $\hat{G}(x,r) = G(x)0$, where r is a single random bit. Then, $\hat{G}$ is perfectly correct, perfectly private and stretch preserving, but its output is not pseudorandom.

Using Lemma 4.5.4, Theorem 4.2.6 and Corollary 4.2.9, we get:

Theorem 4.5.6 *If there exists a pseudorandom generator in $\mathcal{PREN}$ (in particular, in $\oplus$L/poly) then there exists a pseudorandom generator in* NC^0_4.

As in the case of OWF, an adversary that breaks the transformed generator $\hat{G}$ can break, in essentially the same time, the original generator G. Therefore, again, although the new PRG uses extra $m(n)$ random input bits, it is not more secure than the original generator applied to n bits. Moreover, we stress that the PRG $\hat{G}$ one gets from our construction has a sublinear stretch even if G has a large stretch. This follows from the fact that the length $m(n)$ of the random input is typically superlinear in the input length n. However, when G is in NC^0, we can transform it into a PRG $\hat{G}$ in NC^0_4 while (partially) preserving its stretch. Formally,

Theorem 4.5.7 *If there exists a pseudorandom generator with linear stretch in* NC^0 *then there exists a pseudorandom generator with linear stretch in* NC^0_4.

Proof: Let G be a PRG with linear stretch in NC^0. We can apply Theorem 4.2.6 to G and get, by Lemma 4.5.4, a PRG $\hat{G}$ in NC^0_4. We now relate the stretch of $\hat{G}$ to the stretch of G. Let $n, \hat{n}$ be the input complexity of $G, \hat{G}$ (resp.), let $s, \hat{s}$ be the output complexity of $G, \hat{G}$ (resp.), and let $c \cdot n$ be the stretch of G, where c is a constant. The generator $\hat{G}$ is stretch preserving, hence $\hat{s} - \hat{n} = s - n = c \cdot n$. Since G is in NC^0, each of its output bits is computable by a constant size branching program and thus our construction adds only a constant number of random bits for each output bit of G. Therefore, the input length of $\hat{G}$ is linear in the input length of G. Hence, $\hat{s} - \hat{n} = s - n = c \cdot n = \hat{c} \cdot \hat{n}$ for some constant $\hat{c}$ and thus $\hat{G}$ has a linear stretch. ■

Remark 4.5.8 (On the existence of a PRG in $\mathcal{PREN}$) The existence of PRGs in $\mathcal{PREN}$ follows from most standard concrete intractability assumptions. In particular, using Theorem 4.5.6 (applied to PRG collections) one can construct a collection of PRGs in NC^0_4 based on the intractability of factoring [Kha93, NR04] and discrete logarithm [BM84, Yao82]. The existence of PRGs in

$\mathcal{PREN}$ also follows from the existence in $\mathcal{PREN}$ of any *regular* OWF; i.e., a OWF $f = \{f_n\}$ that maps the same (polynomial-time computable) number of elements in $\{0,1\}^n$ to every element in $\mathrm{Im}(f_n)$. (This is the case, for instance, for any one-to-one OWF.) Indeed, the PRG construction from [HILL99] (Theorem 5.4), when applied to a regular OWF f, involves only the computation of universal hash functions and hard-core bits, which can all be implemented in NC^1.[12] Thus a regular OWF in $\mathcal{PREN}$ can be first transformed into a regular OWF in NC^0 and then, using [HILL99], to a PRG in NC^1. Combined with Theorem 4.5.6, this yields a PRG in NC^0_4 based on any regular OWF in $\mathcal{PREN}$.[13] This way, for example, one can construct a (single) PRG in NC^0_4 based on the intractability of lattice problems [HILL99, Ajt96].

Remark 4.5.9 (On unconditional NC^0 reductions from PRG to OWF) Our machinery can be used to obtain an NC^0 reduction from a PRG to any regular OWF (in particular, to any one-to-one OWF), regardless of the complexity of f.[14] Moreover, this reduction only makes a *black-box* use of the underlying regular OWF f (given its regularity parameter $|\mathrm{Im}(f_n)|$). The general idea is to encode the NC^1 construction of [HILL99, Construction 7.1] into a corresponding NC^0 construction. Specifically, suppose $G(x) = g(x, f(q_1(x)), \ldots, f(q_m(x)))$ defines a black-box construction of a PRG G from a OWF f, where g is in $\mathcal{PREN}$ and the q_i's are in NC^0. (The functions $g, q_1, ..., q_m$ are fixed by the reduction and do not depend on f.) Then, letting $\hat{g}((x, y_1, \ldots, y_m), r)$ be a perfect NC^0 encoding of g, the function $\hat{G}(x, r) = \hat{g}((x, f(q_1(x)), \ldots, f(q_m(x))), r)$ perfectly encodes G, and hence defines a black-box NC^0 reduction from a PRG to a OWF. The construction of [HILL99, Construction 7.1] is of the form of $G(x)$ above,[15] assuming that f is regular. Thus, $\hat{G}$ defines an NC^0 reduction from a PRG to a regular OWF.

Comparison with lower bounds. The results of [MST03] rules out the existence of a superlinear-stretch cryptographic PRG in NC^0_4. Thus our NC^0_4 cryptographic PRGs are not far from optimal despite their sublinear stretch. In addition, it is easy to see that there is no PRG with degree 1 or locality 2 (since we can easily decide whether a given string is in the range of such a function). It seems likely that a cryptographic PRG with locality 3 and degree 2 can be constructed (e.g., based on its existence in a higher complexity class), but Theorem 4.5.6 is one step far in terms

[12]In the general case (when the OWF f is not regular) the construction of Håstad et al. (see [HILL99, Construction 7.1]) is not in uniform NC^1, as it requires an additional nonuniform advice of logarithmic length. This (slightly) non-uniform NC^1 construction translates into a *polynomial-time* construction by applying the following steps: (1) construct a polynomial number of PRG candidates (each using a different guess for the non-uniform advice); (2) increase the stretch of each of these candidates using the standard transformation of Goldreich and Micali (cf. [Gol01a, Theorem 3.3.3]); (3) take the exclusive-or of all PRG candidates to obtain the final PRG. The second step requires polynomially many sequential applications of the PRGs, and therefore this construction is not in NC^1. (If we skip the second step the resulting generator will not stretch its input.)

[13] In fact, the same result can be obtained under a relaxed regularity requirement. Specifically, for each n and $y \in \mathrm{Im}(f_n)$ define the value $D_{f,n}(y) = \log |f_n^{-1}(y)|$ and the random variable $R_n = D_{f,n}(f(U_n))$. The NC^1 construction of [HILL99, Construction 7.1] needs to approximate, in poly(n) time, the expectations of both R_n and R_n^2. This is trivially possible when f is regular in the strict sense defined above, since in this case R_n is concentrated on a single (efficiently computable) value. Using a recent NC^1 construction from [HHR06], only the expectation of R_n^2 needs to be efficiently approximated. We finally note that in a non-uniform computation model one can rely on [HILL99] (which gives a nonuniform-NC^1 construction of a PRG from any OWF) and get a PRG in *nonuniform*-NC^0_4 from *any* OWF in $\mathcal{SREN}$.

[14]Viola, in a concurrent work [Vio05], obtains an AC^0 reduction of this type.

[15]The functions $q_1, ..., q_m$ are simply projections there. Interestingly, the recent NC^1 construction from [HHR06] is not of the above form and thus we cannot encode it into an (unconditional) NC^0 construction.

of both locality and degree.[16] See also Table 4.5.1. (These gaps will be partially closed later in this work. Specifically, in Chapter 6 we construct a PRG with locality 4 and *linear* stretch whose security follows from a specific intractability assumption proposed by Alekhnovich in [Ale03], while in Chapter 7 we construct a PRG with locality 3 and degree 2 under the assumption that it is hard to decode a random linear code. Moreover, the latter construction also enjoys an optimal *input* locality.)

4.5.2 ε-Biased Generators

The proof of Lemma 4.5.4 uses the balanced simulator to transform a distinguisher for a PRG G into a distinguisher for its encoding $\hat{G}$. Therefore, if this transformation can be made linear, then the security reduction goes through also in the case of ε-biased generators.

Definition 4.5.10 (ε-biased generator) *An ε-*biased generator *is a polynomial-time computable function, $G : \{0,1\}^n \to \{0,1\}^{l(n)}$, satisfying the following two conditions:*

- **Expansion**: *$l(n) > n$, for all $n \in \mathbb{N}$.*
- **ε-bias**: *For every linear function $L : \{0,1\}^{l(n)} \to \{0,1\}$ and all sufficiently large n's*

$$|\Pr[L(G(U_n)) = 1] - \Pr[L(U_{l(n)}) = 1]| < \varepsilon(n)$$

(where a function L is linear *if its degree over $\mathbb{F}_2$ is 1). By default, the function $\varepsilon(n)$ is required to be negligible.*

Lemma 4.5.11 *Let G be an ε-biased generator and $\hat{G}$ a perfect randomized encoding of G. Assume that the balanced simulator S of $\hat{G}$ is* linear *in the sense that $S(y)$ outputs a randomized linear transformation of y (which is not necessarily a linear function of the simulator's randomness). Then, $\hat{G}$ is also an ε-biased generator.*

Proof: Let $G : \{0,1\}^n \to \{0,1\}^{l(n)}$ and let $\hat{G} : \{0,1\}^n \times \{0,1\}^{m(n)} \to \{0,1\}^{s(n)}$. Assume, towards a contradiction, that $\hat{G}$ is not ε-biased; that is, for some linear function $L : \{0,1\}^{s(n)} \to \{0,1\}$ and infinitely many n's, $|\Pr[L(\hat{G}(U_{n+m})) = 1] - \Pr[L(U_s) = 1]| > \frac{1}{p(n+m)} > \frac{1}{p'(n)}$, where $m = m(n)$, $s = s(n)$, and $p(\cdot), p'(\cdot)$ are polynomials. Using the balance property we get,

$$|\Pr[L(S(G(U_n))) = 1] - \Pr[L(S(U_l)) = 1]| = |\Pr[L(\hat{G}(U_{n+m})) = 1] - \Pr[L(U_s) = 1]| > \frac{1}{p'(n)},$$

where S is the balanced simulator of $\hat{G}$ and the probabilities are taken over the inputs as well as the randomness of S. By an averaging argument we can fix the randomness of S to some string ρ, and get $|\Pr[L(S_\rho(G(U_n))) = 1] - \Pr[L(S_\rho(U_{l(n)})) = 1]| > \frac{1}{p'(n)}$, where S_ρ is the deterministic function defined by using the constant string ρ as the simulator's random input. By the linearity of the simulator, the function $S_\rho : \{0,1\}^l \to \{0,1\}^s$ is linear; therefore the composition of L and S_ρ is also linear, and so the last inequality implies that G is not ε-biased in contradiction to the hypothesis. ■

[16]Note that there exists a PRG with locality 3 if and only if there exists a PRG with degree 2. The "if" direction follows from Lemma 4.2.5 and Lemma 4.5.4, while the "only if" direction follows from Claim 3.3.3 and the fact that each output of an NC^0 PRG must be balanced.

We now argue that the balanced simulators obtained in Section 4.2 are all linear in the above sense. In fact, these simulators satisfy a stronger property: for every fixed random input of the simulator, each bit of the simulator's output is determined by a single bit of its input. This simple structure is due to the fact that we encode non-boolean functions by concatenating the encodings of their output bits. We state here the stronger property as it will be needed in the next subsection.

Observation 4.5.12 *Let S be a simulator of a randomized encoding (of a function) that is obtained by concatenating simulators (i.e., S is defined as in the proof of Lemma 3.2.1). Then, fixing the randomness ρ of S, the simulator's computation has the following simple form: $S_\rho(y) = \sigma_1(y_1)\sigma_2(y_2)\cdots\sigma_l(y_l)$, where each σ_i maps y_i (i.e., the i-th bit of y) to one of two fixed strings. In particular, S computes a randomized degree-1 function of its input.*

Recall that the balanced simulator of the NC^0_4 encoding for functions in $\oplus\mathrm{L}/poly$ (promised by Theorem 4.2.6) is obtained by concatenating the simulators of boolean functions in $\oplus\mathrm{L}/poly$. By Observation 4.5.12, this simulator is linear. Thus, by Lemma 4.5.11, we can construct a sublinear-stretch ε-biased generator in NC^0_4 from any ε-biased generator in $\oplus\mathrm{L}/poly$. In fact, one can easily obtain a nontrivial ε-biased generator even in NC^0_3 by applying the locality construction to each of the bits of the degree-2 generator defined by $G(x, x') = (x, x', \langle x, x'\rangle)$, where $\langle\cdot,\cdot\rangle$ denotes inner product modulo 2. Again, the resulting encoding is obtained by concatenation and thus, by Observation 4.5.12 and Lemma 4.5.11, is also ε-biased. (This generator actually fools a much larger class of statistical tests; see Section 4.5.3 below.) Thus, we have:

Theorem 4.5.13 *There is a (sublinear-stretch) ε-biased generator in* NC^0_3.

Building on a construction of Mossel et al., it is in fact possible to achieve linear stretch in NC^0_3. Namely,

Theorem 4.5.14 *There is a linear-stretch ε-biased generator in* NC^0_3.

Proof: Mossel et al. present an ε-biased generator in NC^0 with degree 2 and linear stretch ([MST03], Theorem 13).[17] Let G be their ε-biased generator. We can apply the locality construction (4.2.4) to G (using concatenation) and get, by Lemma 4.5.11 and Observation 4.5.12, an ε-biased generator $\hat{G}$ in NC^0_3. We now relate the stretch of $\hat{G}$ to the stretch of G. Let $n, \hat{n}$ be the input complexity of $G, \hat{G}$ (resp.), let $s, \hat{s}$ be the output complexity of $G, \hat{G}$ (resp.), and let $c \cdot n$ be the stretch of G, where c is a constant. The generator $\hat{G}$ is stretch preserving, hence $\hat{s} - \hat{n} = s - n = c \cdot n$. Since G is in NC^0, each of its output bits can be represented as a polynomial that has a constant number of monomials and thus the locality construction adds only a constant number of random bits for each output bit of G. Therefore, the input length of $\hat{G}$ is linear in the input length of G. Hence, $\hat{s} - \hat{n} = s - n = c \cdot n = \hat{c} \cdot \hat{n}$ for some constant $\hat{c}$ and thus $\hat{G}$ has a linear stretch. ■

[17] In fact, the generator of [MST03, Theorem 13] is in nonuniform-NC^0_5 (and it has a slightly superlinear stretch). However, a similar construction gives an ε-biased generator in *uniform* NC^0 with degree 2 and linear stretch. (The locality of this generator is large but constant.) This can be done by replacing the probabilistic construction given in [MST03, Lemma 12] with a uniform construction of constant-degree bipartite expander with some "good" expansion properties – such a construction is given in [CRVW02, Theorem 7.1].

Comparison with lower bounds. It is not hard to see that there is no ε-biased generator with degree 1 or locality 2.[18] In [CM01] it was shown that there is no superlinear-stretch ε-biased generator in NC^0_3. Thus, our linear-stretch NC^0_3 generator (building on the one from [MST03]) is not only optimal with respect to locality and degree but is also essentially optimal with respect to stretch.

4.5.3 Generators for Space-Bounded Computation

We turn to the case of PRGs for space-bounded computation. A standard way of modeling a randomized space-bounded Turing machine is by having a random tape on which the machine can access the random bits one by one but cannot "go back" and view previous random bits (i.e., any bit that the machine wishes to remember, it must store in its limited memory). For the purpose of derandomizing such machines, it suffices to construct PRGs that fool any space-bounded distinguisher having a similar one-way access to its input. Following Babai et al. [BNS89], we refer to such distinguishers as *space-bounded distinguishers.*

Definition 4.5.15 ([BNS89]) (Space-bounded distinguisher) *A* space-$s(n)$ distinguisher *is a deterministic Turing machine M, and an infinite sequence of binary strings $a = (a_1, \dots, a_n, \dots)$ called the advice strings, where $|a_n| = 2^{O(s(n))}$. The machine has the following tapes: read-write work tapes, a read-only advice tape, and a read-only input tape on which the tested input string, y, is given. The input tape has a one-way mechanism to access the tested string; namely, at any point it may request the next bit of y. In addition, only $s(n)$ cells of the work tapes can be used. Given an n-bit input, y, the output of the distinguisher, $M^a(y)$, is the (binary) output of M where y is given on the input tape and a_n is given on the advice tape.*

This class of distinguishers is a proper subset of the distinguishers that can be implemented by a space-$s(n)$ Turing machine with a two-way access to the input. Nevertheless, even log-space distinguishers are quite powerful, and many distinguishers fall into this category. In particular, this is true for the class of *linear* distinguishers considered in Section 4.5.2.

Definition 4.5.16 (PRG for space-bounded computation) *We say that a polynomial-time computable function $G : \{0,1\}^n \to \{0,1\}^{l(n)}$ is a PRG for space $s(n)$ if $l(n) > n$ and $G(U_n)$ is indistinguishable from $U_{l(n)}$ to any space-$s(n)$ distinguisher. That is, for every space-$s(n)$ distinguisher M^a, the distinguishing advantage $|\Pr[M^a(G(U_n)) = 1] - \Pr[M^a(U_{l(n)}) = 1]|$ is negligible in n.*

Several constructions of high-stretch PRGs for space-bounded computation exist in the literature (e.g., [BNS89, Nis92]). In particular, a PRG for logspace computation from [BNS89] can be computed using logarithmic space, and thus, by Theorem 4.2.6, admits an efficient perfect encoding in NC^0_4. It can be shown (see proof of Theorem 4.5.17) that this NC^0_4 encoding fools logspace distinguishers as well; hence, we can reduce the security of the randomized encoding to the security of the encoded generator, and get an NC^0_4 PRG that fools logspace computation. However, as in the case of ε-biased generators, constructing such PRGs with a low stretch is much easier. In fact, the same "inner product" generator we used in Section 4.5.2 can do here is well.

[18] A degree 1 generator contains more than n linear functions over n variables, which must be linearly dependent and thus biased. The non-existence of a 2-local generator follows from the fact that every nonlinear function of two input bits is biased.

Theorem 4.5.17 *There exists a (sublinear-stretch) PRG for sublinear-space computation in* NC^0_3.

Proof: Consider the inner product generator $G(x,x') = (x, x', \langle x, x'\rangle)$, where $x, x' \in \{0,1\}^n$. It follows from the average-case hardness of the inner product function for two-party communication complexity [CG88] that G fools all sublinear-space distinguishers. (Indeed, a sublinear-space distinguisher implies a sublinear-communication protocol predicting the inner product of x and x'. Specifically, the party holding x runs the distinguisher until it finishes reading x, and then sends its configuration to the party holding x'.)

Applying the locality construction to G, we obtain a perfect encoding $\hat{G}$ in NC^0_3. (In fact, we can apply the locality construction only to the last bit of G and leave the other outputs as they are.) We argue that $\hat{G}$ inherits the pseudorandomness of G. As before, we would like to argue that if $\hat{M}$ is a sublinear-space distinguisher breaking $\hat{G}$ and S is the balanced simulator of the encoding, then $\hat{M}(S(\cdot))$ is a sublinear-space distinguisher breaking G. Similarly to the proof of Lemma 4.5.11, the fact that $\hat{M}(S(\cdot))$ can be implemented in sublinear space will follow from the simple structure of S. However, in contrast to Lemma 4.5.11, here it does not suffice to require S to be linear and we need to rely on the stronger property guaranteed by Observation 4.5.12.[19]

We now formalize the above. As argued in Observation 4.5.12, fixing the randomness ρ of S, the simulator's computation can be written as $S_\rho(y) = \sigma_1(y_1)\sigma_2(y_2)\cdots\sigma_l(y_l)$, where each σ_i maps a bit of y to one of two fixed strings. We can thus use S to turn a sublinear-space distinguisher $\hat{M}^a$ breaking $\hat{G}$ into a sublinear-space distinguisher $M^{a'}$ breaking G. Specifically, let the advice a' include, in addition to a, the $2l$ strings $\sigma_i(0), \sigma_i(1)$ corresponding to a "good" ρ which maintains the distinguishing advantage. (The existence of such ρ follows from an averaging argument.) The machine $M^{a'}(y)$ can now emulate the computation of $\hat{M}^a(S_\rho(y))$ using sublinear space and a one-way access to y by applying $\hat{M}^a$ in each step to the corresponding string $\sigma_i(y_i)$. ■

4.5.4 Pseudorandom Generators — Conclusion

We conclude this section with Table 4.5.1, which summarizes some of the PRGs constructed here as well as previous ones from [MST03] and highlights the remaining gaps. We will partially close these gaps in Chapters 6 and 7 under specific *intractability* assumptions. (In particular, we will construct a PRG with locality 4 and *linear* stretch, as well as a PRG with output locality 3, input locality 3, and degree 2.)

4.6 Collision-Resistant Hashing in NC^0

We start with a formal definition of collision-resistant hash-functions (CRHFs).

Definition 4.6.1 (Collision-resistant hashing) *Let* $\ell, \ell' : \mathbb{N} \to \mathbb{N}$ *be such that* $\ell(n) > \ell'(n)$ *and let* $Z \subseteq \{0,1\}^*$. *A collection of functions* $\{h_z\}_{z\in Z}$ *is said to be* collision-resistant *if the following holds:*

[19] Indeed, in the current model of (non-uniform) space-bounded computation with *one-way* access to the input (and two-way access to the advice), there exist a boolean function $\hat{M}$ computable in sublinear space and a linear function S such that the composed function $\hat{M}(S(\cdot))$ is not computable in sublinear space. For instance, let $\hat{M}(y_1,\ldots,y_{2n}) = y_1y_2 + y_3y_4 + \ldots + y_{2n-1}y_{2n}$ and $S(x_1,\ldots,x_{2n}) = (x_1, x_{n+1}, x_2, x_{n+2}, \ldots, x_n, x_{2n})$.

Type	Stretch	Locality	Degree
ε-biased	superlinear	5	2 ✓
ε-biased	$n^{\Omega(\sqrt{k})}$	large k	$\Omega(\sqrt{k})$
ε-biased	$\Omega(n^2)$✓	$\Omega(n)$	2 ✓
ε-biased	linear ✓	3 ✓	2 ✓
space	sublinear $\not\checkmark$	3 ✓	2 ✓
cryptographic *	sublinear $\not\checkmark$	4	3

Table 4.5.1: Summary of known pseudorandom generators. Results of Mossel et al. [MST03] appear in the top part and results of this paper in the bottom part. A parameter is marked as optimal (✓) if when fixing the other parameters it cannot be improved. A stretch entry is marked with $\not\checkmark$ if the stretch is sublinear and cannot be improved to be superlinear (but might be improved to be linear). The symbol * indicates a conditional result.

1. *There exists a probabilistic polynomial-time* key-generation *algorithm, G, that on input 1^n outputs an* index *$z \in Z$ (of a function h_z). The function h_z maps strings of length $\ell(n)$ to strings of length $\ell'(n)$.*

2. *There exists a polynomial-time* evaluation *algorithm that on input $z \in G(1^n), x \in \{0,1\}^{\ell(n)}$ computes $h_z(x)$.*

3. *Collisions are hard to find. Formally, a pair (x, x') is called a* collision *for a function h_z if $x \neq x'$ but $h_z(x) = h_z(x')$. The collision-resistance requirement states that every non-uniform polynomial-time algorithm A, that is given input $(z = G(1^n), 1^n)$, succeeds in finding a collision for h_z with a negligible probability in n (where the probability is taken over the coin tosses of both G and A).*

Lemma 4.6.2 *Suppose $\mathcal{H} = \{h_z\}_{z \in Z}$ is collision resistant and $\hat{\mathcal{H}} = \{\hat{h}_z\}_{z \in Z}$ is a perfect randomized encoding of $\mathcal{H}$. Then $\hat{\mathcal{H}}$ is also collision resistant.*

Proof: Since $\hat{h}_z$ is stretch preserving, it is guaranteed to shrink its input as h_z. The key generation algorithm G of $\mathcal{H}$ is used as the key generation algorithm of $\hat{\mathcal{H}}$. By the uniformity of the collection $\hat{\mathcal{H}}$, there exists an efficient evaluation algorithm for this collection. Finally, any collision $((x, r), (x', r'))$ under $\hat{h}_z$ (i.e., $(x, r) \neq (x', r')$ and $\hat{h}_z(x, r) = \hat{h}_z(x', r')$), defines a collision (x, x') under h_z. Indeed, perfect correctness ensures that $h_z(x) = h_z(x')$ and unique-randomness (see Lemma 3.2.5) ensures that $x \neq x'$. Thus, an efficient algorithm that finds collisions for $\hat{\mathcal{H}}$ with non-negligible probability yields a similar algorithm for $\mathcal{H}$. ∎

By Lemma 4.6.2, Theorem 4.2.6 and Corollary 4.2.9, we get:

Theorem 4.6.3 *If there exists a CRHF $\mathcal{H} = \{h_z\}_{z \in Z}$ such that the function $h'(z, x) \overset{def}{=} h_z(x)$ is in $\mathcal{PREN}$ (in particular, in $\oplus$L/poly), then there exists a CRHF $\hat{\mathcal{H}} = \{\hat{h}_z\}_{z \in Z}$ such that the mapping $(z, y) \mapsto \hat{h}_z(y)$ is in NC^0_4.*

Using Theorem 4.6.3, we can construct CRHFs in NC^0 based on the intractability of factoring [Dam88], discrete logarithm [Ped91], or lattice problems [GGH96, Reg03]. All these candidates are computable in NC^1 provided that some pre-computation is done by the key-generation algorithm. Note that the key generation algorithm of the resulting NC^0 CRHF is not in NC^0. For more details on NC^0 computation of collections of cryptographic primitives see Appendix A.

4.7 Encryption in NC^0

We turn to the case of encryption. We first show that computational encoding preserves the security of semantically-secure encryption scheme both in the public-key and in the private-key setting. This result provides an NC^0 encryption algorithm but does not promise anything regarding the parallel complexity of the decryption process. This raises the question whether decryption can also be implemented in NC^0. In Section 4.7.2 we show that, except for some special settings (namely, private-key encryption which is secure only for single message of bounded length or stateful private-key encryption), decryption in NC^0 is impossible regardless of the complexity of encryption. Finally, in Section 4.7.3 we explore the effect of randomized encoding on encryption schemes which enjoy stronger notions of security. In particular, we show that randomized encoding preserves security against chosen plaintext attacks (CPA) as well as a-priory chosen ciphertext attacks (CCA1). However, randomized encoding does not preserve security against a-posteriori chosen ciphertext attack (CCA2). Still, we show that the encoding of a CCA2-secure scheme enjoys a relaxed security property that suffices for most applications of CCA2-security.

4.7.1 Main Results

Suppose that $\mathcal{E} = (G, E, D)$ is a public-key encryption scheme, where G is a key generation algorithm, the encryption function $E(e, x, r)$ encrypts the message x using the key e and randomness r, and $D(d, y)$ decrypts the cipher y using the decryption key d. As usual, the functions G, E, D are polynomial-time computable, and the scheme provides correct decryption and satisfies indistinguishability of encryptions [GM84b]. Let $\hat{E}$ be a randomized encoding of E, and let $\hat{D}(d, \hat{y}) \stackrel{\text{def}}{=} D(d, C(\hat{y}))$ be the composition of D with the decoder B of $\hat{E}$. We argue that the scheme $\hat{\mathcal{E}} \stackrel{\text{def}}{=} (G, \hat{E}, \hat{D})$ is also a public-key encryption scheme. The efficiency and correctness of $\hat{\mathcal{E}}$ are guaranteed by the uniformity of the encoding and its correctness. Using the efficient simulator of $\hat{E}$, we can reduce the security of $\hat{\mathcal{E}}$ to that of $\mathcal{E}$. Namely, given an efficient adversary $\hat{A}$ that distinguishes between encryptions of x and x' under $\hat{\mathcal{E}}$, we can break $\mathcal{E}$ by using the simulator to transform original ciphers into "new" ciphers, and then invoke $\hat{A}$. The same argument holds in the private-key setting. We now formalize this argument.

Definition 4.7.1 (Public-key encryption) *A* secure public-key encryption scheme *(PKE) is a triple (G, E, D) of probabilistic polynomial-time algorithms satisfying the following conditions:*

- ***Viability**: On input 1^n the key generation algorithm, G, outputs a pair of keys (e, d). For every pair (e, d) such that $(e, d) \in G(1^n)$, and for every plaintext $x \in \{0, 1\}^*$, the algorithms E, D satisfy*

  $$\Pr[D(d, E(e, x)) \neq x)] \leq \varepsilon(n)$$

  where $\varepsilon(n)$ is a negligible function and the probability is taken over the internal coin tosses of algorithms E and D.

- ***Security**: For every polynomial $\ell(\cdot)$, and every families of plaintexts $\{x_n\}_{n\in\mathbb{N}}$ and $\{x'_n\}_{n\in\mathbb{N}}$ where $x_n, x'_n \in \{0,1\}^{\ell(n)}$, it holds that*

$$(e \leftarrow G_1(1^n), E(e, x_n)) \overset{c}{\equiv} (e \leftarrow G_1(1^n), E(e, x'_n)), \tag{4.7.1}$$

where $G_1(1^n)$ denotes the first element in the pair $G(1^n)$.

The definition of a *private-key* encryption scheme is similar, except that the public key is omitted from the ensembles. That is, instead of Equation 4.7.1 we require that $E(G_1(1^n), x_n) \overset{c}{\equiv} E(G_1(1^n), x'_n)$. An extension to multiple-message security, where the indistinguishability requirement should hold for encryptions of polynomially many messages, follows naturally (see [Gol04, chapter 5] for formal definitions). In the public-key case, multiple-message security is implied by single-message security as defined above, whereas in the private-key case it is a strictly stronger notion. In the following we explicitly address only the (single-message) public-key case, but the treatment easily holds for the case of private-key encryption with multiple-message security.

Lemma 4.7.2 *Let $\mathcal{E} = (G, E, D)$ be a secure public-key encryption scheme, where $E(e, x, r)$ is viewed as a polynomial-time computable function that encrypts the message x using the key e and randomness r. Let $\hat{E}((e, x), (r, s)) = \hat{E}((e, x, r), s)$ be a computational randomized encoding of E and let $\hat{D}(d, \hat{y}) \overset{def}{=} D(d, B(\hat{y}))$ be the composition of D with the decoder B of $\hat{E}$. Then, the scheme $\hat{\mathcal{E}} \overset{def}{=} (G, \hat{E}, \hat{D})$ is also a secure public-key encryption scheme.*

Proof: The uniformity of the encoding guarantees that the functions $\hat{E}$ and $\hat{D}$ can be efficiently computed. The viability of $\hat{\mathcal{E}}$ follows in a straightforward way from the correctness of the decoder B. Indeed, if (e, d) are in the support of $G(1^n)$, then for any plaintext x we have

$$\begin{aligned}
\Pr_{r,s}[\hat{D}(d, \hat{E}(e, x, r, s)) \neq x] &= \Pr_{r,s}[D(d, B(\hat{E}(e, x, r, s))) \neq x] \\
&\leq \Pr_{r,s}[B(\hat{E}((e, x, r), s)) \neq E(e, x, r)] + \Pr_{r}[D(d, E(e, x, r)) \neq x] \\
&\leq \varepsilon(n),
\end{aligned}$$

where $\varepsilon(\cdot)$ is negligible in n and the probabilities are also taken over the coin tosses of D; the first inequality follows from the union bound and the second from the viability of $\mathcal{E}$ and the statistical correctness of $\hat{E}$.

We move on to prove the security of the construction. Let S be the efficient computational simulator of $\hat{E}$. Then, for every polynomial $\ell(\cdot)$, and every families of plaintexts $\{x_n\}_{n\in\mathbb{N}}$ and $\{x'_n\}_{n\in\mathbb{N}}$ where $x_n, x'_n \in \{0,1\}^{\ell(n)}$, it holds that

$$\begin{aligned}
(e \leftarrow G_1(1^n), \hat{E}(e, x_n, r_n, s_n)) &\overset{c}{\equiv} (e \leftarrow G_1(1^n), S(E(e, x_n, r_n))) && \text{(by the privacy of } \hat{E}\text{, Fact 2.2.10)} \\
&\overset{c}{\equiv} (e \leftarrow G_1(1^n), S(E(e, x'_n, r_n))) && \text{(by the security of } \mathcal{E}\text{, Fact 2.2.8)} \\
&\overset{c}{\equiv} (e \leftarrow G_1(1^n), \hat{E}(e, x'_n, r_n, s_n)) && \text{(by the privacy of } \hat{E}\text{, Fact 2.2.10),}
\end{aligned}$$

where r_n and s_n are uniformly chosen random strings of an appropriate length. Hence, the security of $\hat{\mathcal{E}}$ follows from the transitivity of the relation $\overset{c}{\equiv}$ (Fact 2.2.6). ■

In particular, if the scheme $\mathcal{E} = (G, E, D)$ enables errorless decryption and the encoding $\hat{E}$ is perfectly correct, then the scheme $\hat{\mathcal{E}}$ also enables errorless decryption. Additionally, the above lemma is easily extended to case of private-key encryption with multiple-message security. Thus we get,

Theorem 4.7.3 *If there exists a secure public-key encryption scheme (respectively, a secure private-key encryption scheme) $\mathcal{E} = (G, E, D)$, such that E is in $\mathcal{CREN}$ (in particular, in* NL*/poly or* $\oplus$L*/poly), then there exists a secure public-key encryption scheme (respectively, a secure private-key encryption scheme) $\hat{\mathcal{E}} = (G, \hat{E}, \hat{D})$, such that $\hat{E}$ is in* NC^0_4.

Specifically, one can construct an NC^0 PKE based on either factoring [Rab79, GL89, BG85], the Diffie-Hellman Assumption [Gam85, GL89] or lattice problems [AD97, Reg03]. (These schemes enable an NC^1 encryption algorithm given a suitable representation of the key.)

4.7.2 On Decryption in NC^0

Our construction provides an NC^0 encryption algorithm but does not promise anything regarding the parallel complexity of the decryption process. This raises the question whether decryption can also be implemented in NC^0.

Negative results

We now show that, in many settings, decryption in NC^0 is impossible regardless of the complexity of encryption. Here we consider standard *stateless* encryption schemes. We begin with the case of multiple-message security (in either the private-key or public-key setting). If a decryption algorithm $D(d, y)$ is in NC^0_k, then an adversary that gets n encrypted messages can correctly guess the first bits of *all* the plaintexts (jointly) with at least 2^{-k} probability. To do so, the adversary simply guesses at random the k (or less) bits of the key d on which the first output bit of D depends, and then computes this first output bit (which is supposed to be the first plaintext bit) on each of the n ciphertexts using the subkey it guessed. Whenever the adversary guesses the k bits correctly, it succeeds to find the first bits of *all* n messages. When $n > k$, this violates the semantic security of the encryption scheme. Indeed, for the encryption scheme to be secure, the adversary's success probability (when the messages are chosen at random) can only be negligibly larger than 2^{-n}. (That is, an adversary cannot do much better than simply guessing these first bits.)

Even in the case of a single-message private-key encryption, it is impossible to implement decryption in NC^0_k with an arbitrary (polynomial) message length. Indeed, when the message length exceeds $(2|d|)^k$ (where $|d|$ is the length of the decryption key), there must be more than 2^k bits of the output of D which depend on the same k bits of the key, in which case we are in the same situation as before. That is, we can guess the value of more than 2^k bits of the message with constant success probability 2^{-k}. Again, if we consider a randomly chosen message, this violates semantic security.

Positive results

In contrast to the above, if the scheme is restricted to a *single* message of a bounded length (even larger than the key) we can use our machinery to construct a private-key encryption scheme in which both encryption and decryption can be computed in NC^0. This can be done by using the output of an NC^0 PRG to mask the plaintext. Specifically, let $E(e, x) = G(e) \oplus x$ and $D(e, y) = y \oplus G(e)$, where e is a uniformly random key generated by the key generation algorithm and G is a PRG. Unfortunately, the resulting scheme is severely limited by the low stretch of our PRGs. This approach can be also used to give multiple message security, at the price of requiring the encryption and decryption algorithms to maintain a synchronized *state*. In such a stateful encryption scheme the encryption and decryption algorithms take an additional input and produce an additional output, corresponding to their state before and after the operation. The seed of the generator can be used, in this case, as the state of the scheme. In this setting, we can obtain multiple-message security by refreshing the seed of the generator in each invocation; e.g., when encrypting the current bit the encryption algorithm can randomly choose a new seed for the next session, encrypt it along with current bit, and send this encryption to the receiver (alternatively, see [Gol04, Construction 5.3.3]). In the resulting scheme both encryption and decryption are NC^0 functions whose inputs include the inner state of the algorithm.

4.7.3 Security against CPA, CCA1 and CCA2 Attacks

In this section we address the possibility of applying our machinery to encryption schemes that enjoy stronger notions of security. In particular, we consider schemes that are secure against chosen plaintext attacks (CPA), a-priory chosen ciphertext attacks (CCA1), and a-posteriori chosen ciphertext attacks (CCA2). In all three attacks the adversary has to win the standard indistinguishability game (i.e., given a ciphertext $c = E(e, m_b)$ find out which of the two predefined plaintexts m_0, m_1 was encrypted), and so the actual difference lies at the power of the adversary. In a CPA attack the adversary can obtain encryptions of plaintexts of his choice (under the key being attacked), i.e., the adversary gets an oracle access to the encryption function. In CCA1 attack the adversary may also obtain decryptions of his choice (under the key being attacked), but he is allowed to do so only *before* the challenge is presented to him. In both cases, the security is preserved under randomized encoding. We briefly sketch the proof idea.

Let $\hat{A}$ be an adversary that breaks the encoding $\hat{\mathcal{E}}$ via a CPA attack (resp. CCA1 attack). We use $\hat{A}$ to obtain an adversary A that breaks the original scheme $\mathcal{E}$. As in the proof of Lemma 4.7.2, A uses the simulator to translate the challenge c, an encryption of the message m_b under $\mathcal{E}$, into a challenge $\hat{c}$, which is an encryption of the same message under $\hat{\mathcal{E}}$. Similarly, A answers the encryption queries of $\hat{A}$ (to the oracle $\hat{E}$) by directing these queries to the oracle E and applying the simulator to the result. Also, in the case of CCA1 attack, whenever $\hat{A}$ asks the decryption oracle $\hat{D}$ to decrypt some ciphertext $\hat{c}'$, the adversary A uses the decoder (of the encoding) to translate $\hat{c}'$ into a ciphertext c' of the same message under the scheme $\mathcal{E}$, and then uses the decryption oracle D to decrypt c'. This allows A to emulate the oracles $\hat{D}$ and $\hat{E}$, and thus to translate a successful CPA attack (resp. CCA1 attack) on the new scheme into a similar attack on the original scheme.

The situation is different in the case of a CCA2 attack. As in the case of a CCA1 attack, a CCA2 attacker has an oracle access to the decryption function corresponding to the decryption key in use; however, the adversary can query the oracle *even after* the challenge has been given to him, under the restriction that he cannot ask the oracle to decrypt the challenge c itself.

We start by observing that when applying a randomized encoding to a CCA2-secure encryption scheme, CCA2 security may be lost. Indeed, in the resulting encryption one can easily modify a given ciphertext challenge $\hat{c} = \hat{E}(e, x, r)$ into a ciphertext $\hat{c}' \neq \hat{c}$ which is also an encryption of the same message under the same encryption key. This can be done by applying the decoder (of the randomized encoding $\hat{E}$) and then the simulator on $\hat{c}$, that is $\hat{c}' = S(C(\hat{c}))$. Hence, one can break the encryption by simply asking the decryption oracle to decrypt $\hat{c}'$.

It is instructive to understand why the previous arguments fail to generalize to the case of CCA2 security. In the case of CCA1 attacks we transformed an adversary $\hat{A}$ that breaks the encoding $\hat{\mathcal{E}}$ into an adversary A for the original scheme in the following way: (1) we used the simulator to convert a challenge $c = E(e, m_b)$ into a challenge $\hat{c}$ which is an encryption of the same message under $\hat{\mathcal{E}}$; (2) when $\hat{A}$ asks $\hat{D}$ to decrypt a ciphertext $\hat{c}'$, the adversary A uses the decoder (of the encoding) to translate $\hat{c}'$ into a ciphertext c' of the same message under the scheme $\mathcal{E}$, and then asks the decryption oracle D to decrypt c'. However, recall that in a CCA2 attack the adversaries are not allowed to ask the oracle to decrypt the challenge itself (after the challenge is presented). So if $c' = c$ but $\hat{c}' \neq \hat{c}$, the adversary A cannot answer the (legitimate) query of $\hat{A}$.

To complement the above, we show that when applying a randomized encoding to a CCA2-secure encryption scheme not all is lost. Specifically, the resulting scheme still satisfies *Replayable CCA security (RCCA)*, a relaxed variant of CCA2 security that was suggested in [CKN03]. Loosely speaking, RCCA security captures encryption schemes that are CCA2 secure except that they allow anyone to generate new ciphers that decrypt to the same value as a given ciphertext. More precisely, an RCCA attack is a CCA2 attack in which the adversary cannot ask the oracle to decrypt *any* cipher c' that decrypts to either m_0 or m_1 (cf. [CKN03, Figure 3]). This limitation prevents the problem raised in the CCA2 proof, in which a legitimate query for $\hat{D}$ translates by the decoder into an illegitimate query for D. That is, if $\hat{c}'$ does not decrypt under $\hat{\mathcal{E}}$ to neither m_0 nor m_1, then (by correctness) the ciphertext c' obtained by applying the decoder to $\hat{c}'$ does not decrypt to any of these messages either. Hence, randomized encoding preserves RCCA security. As argued in [CKN03], RCCA security suffices in most applications of CCA2 security.

4.8 Other Cryptographic Primitives

The construction that was used for encryption can be adapted to other cryptographic primitives including (non-interactive) commitments, signatures, message authentication schemes (MACs), and non-interactive zero-knowledge proofs (for definitions see [Gol01a, Gol04]). In all these cases, we can replace the sender (i.e., the encrypting party, committing party, signer or prover, according to the case) with its randomized encoding and let the receiver (the decrypting party or verifier) use the decoding algorithm to translate the output of the new sender to an output of the original one. The security of the resulting scheme reduces to the security of the original one by using the efficient simulator and decoder. In fact, such a construction can also be generalized to the case of interactive protocols such as zero-knowledge proofs and interactive commitments. As in the case of encryption discussed above, this transformation results in an NC^0 sender but does not promise anything regarding the parallel complexity of the receiver. (In all these cases, we show that it is impossible to implement the receiver in NC^0.) An interesting feature of the case of commitment is that we can also improve the parallel complexity at the receiver's end (see below). The same holds for applications of commitment such as coin-flipping and ZK proofs. We now briefly sketch these constructions and their security proofs.

4.8.1 Signatures

Let $\mathcal{S} = (G, S, V)$ be a signature scheme, where G is a key-generation algorithm that generates the signing and verification keys (s, v), the signing function $S(s, \alpha, r)$ computes a signature β on the document α using the key s and randomness r, and the verification algorithm $V(v, \alpha, \beta)$ verifies that β is a valid signature on α using the verification key v. The three algorithms run in probabilistic polynomial time, and the scheme provides correct verification for legal signatures (ones that were produced by the signing function using the corresponding signing key). The scheme is secure (unforgeable) if it is infeasible to forge a signature in a chosen message attack. Namely, any (non-uniform) polynomial-time adversary that gets the verification key and an oracle access to the signing process $S(s, \cdot)$ fails to produce a valid signature β on a document α (with respect to the corresponding verification key v) for which it has not requested a signature from the oracle. (When the signing algorithm is probabilistic, the attacker does not have an access to the random coin tosses of the signing algorithm.)

Let $\hat{S}$ be a computational randomized encoding of S, and let $\hat{V}(v, \alpha, \hat{\beta}) \stackrel{\text{def}}{=} V(v, \alpha, B(\hat{\beta}))$ be the composition of V with the decoder B of the encoding $\hat{S}$. We claim that the scheme $\hat{\mathcal{S}} \stackrel{\text{def}}{=} (G, \hat{S}, \hat{V})$ is also a signature scheme. The efficiency and correctness of $\hat{\mathcal{S}}$ follow from the uniformity of the encoding and its correctness. To prove the security of the new scheme we use the simulator to transform an attack on $\hat{\mathcal{S}}$ into an attack on $\mathcal{S}$. Specifically, given an adversary $\hat{A}$ that breaks $\hat{\mathcal{S}}$, we can break $\mathcal{S}$ by invoking $\hat{A}$ and emulating the oracle $\hat{S}$ using the simulator of the encoding and the signature oracle S. By Fact 2.2.9, the output of $\hat{A}$ when interacting with the emulated oracle is computationally indistinguishable from its output when interacting with the actual signing oracle $\hat{S}(s, \cdot)$. Moreover, if the forged signature $(\alpha, \hat{\beta})$ produced by $\hat{A}$ is valid under $\hat{\mathcal{S}}$, then it is translated into a valid signature (α, β) under $\mathcal{S}$ by using the decoder, i.e., $\beta = B(\hat{\beta})$. Hence, if the scheme $\hat{\mathcal{S}}$ can be broken with non-negligible probability, then so can the scheme $\mathcal{S}$. A similar argument holds also in the private-key setting (i.e., in the case of MACs).[20] We will later (Remark 5.3.3) show that signatures and MACs whose signing algorithm is in NC^0_4 can be based on the intractability of factoring, the discrete logarithm problem, and lattice problems.

Impossibility of NC^0 verification. It is not hard to see that the verification algorithm V cannot be realized in NC^0. Indeed, if $V \in \mathrm{NC}^0_k$ one can forge a signature on any document α by simply choosing a random string β' of an appropriate length. This attack succeeds with probability 2^{-k} since: (1) the verification algorithm checks the validity of the signature by reading at most k bits of the signature; and (2) the probability that β' agrees with the correct signature β on the bits which are read by V is at least 2^{-k}.

4.8.2 Commitments

A commitment scheme enables one party (a sender) to commit itself to a value while keeping it secret from another party (the receiver). Later, the sender can reveal the committed value to the receiver, and it is guaranteed that the revealed value is equal to the one determined at the commit stage.

Non-interactive commitments

We start with the simple case of a perfectly binding, non-interactive commitment. Such a scheme can be defined by a polynomial-time computable function $\textsc{Send}(b, r)$ that outputs a commitment c to the bit b using the randomness r. We assume, w.l.o.g., that the scheme has a canonical decommit stage in which the sender reveals b by sending b and r to the receiver, who verifies that $\textsc{Send}(b, r)$ is equal to the commitment c. The scheme should be both (computationally) hiding and (perfectly) binding. Hiding requires that $c = \textsc{Send}(b, r)$ keeps b computationally secret, that is $\textsc{Send}(0, U_n) \stackrel{c}{\equiv}$

[20]Our NC^0 signing algorithm is probabilistic but this is unavoidable. Indeed, while a signing algorithm may generally be deterministic (see [Gol04, p. 506]), an NC^0 signing algorithm cannot be deterministic as in this case an adversary can efficiently learn it and use it to forge messages.

$\textsc{Send}(1, U_n)$. Binding means that it is impossible for the sender to open its commitment in two different ways; that is, there are no r_0 and r_1 such that $\textsc{Send}(0, r_0) = \textsc{Send}(1, r_1)$.

Let $\widehat{\textsc{Send}}(b, r, s)$ be a perfectly-correct computationally-private encoding of $\textsc{Send}(b, r)$. Then $\widehat{\textsc{Send}}$ defines a computationally-hiding perfectly-binding, non-interactive commitment. Hiding follows from the privacy of the encoding, as argued for the case of encryption in Lemma 4.7.2. Namely, it holds that

$$\widehat{\textsc{Send}}(0, r, s) \overset{c}{\equiv} S(\textsc{Send}(0, r, s)) \overset{c}{\equiv} S(\textsc{Send}(1, r, s)) \overset{c}{\equiv} \widehat{\textsc{Send}}(1, r, s)$$

where r and s are uniformly chosen strings of an appropriate length (the first and third transitions follow from the privacy of $\widehat{\textsc{Send}}$ and Fact 2.2.10, while the second transition follows from the hiding of $\textsc{Send}$ and Fact 2.2.8). The binding property of $\widehat{\textsc{Send}}$ follows from the perfect correctness; namely, if there exists an ambiguous pair $(r_0, s_0), (r_1, s_1)$ such that $\widehat{\textsc{Send}}(0, r_0, s_0) = \widehat{\textsc{Send}}(1, r_1, s_1)$, then by perfect correctness it holds that $\textsc{Send}(0, r_0) = \textsc{Send}(1, r_1)$ which contradicts the binding of the original scheme.[21] So when the encoding is in NC^0 we get a commitment scheme whose sender is in NC^0.

In fact, in contrast to the primitives described so far, here we also improve the parallel complexity at the receiver's end. Indeed, on input $\hat{c}, b, r, s$ the receiver's computation consists of computing $\widehat{\textsc{Send}}(b, r, s)$ and comparing the result to $\hat{c}$. Assuming $\widehat{\textsc{Send}}$ is in NC^0, the receiver can be implemented by an NC^0 circuit augmented with a single (unbounded fan-in) AND gate. We refer to this special type of AC^0 circuit as an $\mathrm{AND}_n \circ \mathrm{NC}^0$ circuit. This extension of NC^0 is necessary as the locality of the function f that the receiver computes cannot be constant. (See the end of this subsection.)

Remark 4.8.1 (Unconditional NC^0 construction of non-interactive commitment from 1-1 OWF) We can use our machinery to obtain an unconditional NC^0 reduction from a non-interactive commitment scheme to any one-to-one OWF. Moreover, this reduction only makes a *black-box* use of the underlying OWF f. As in the case of the NC^0 reduction from PRGs to regular OWFs (Remark 4.5.9), the idea is to encode a non-adaptive black-box NC^1 reduction into a corresponding NC^0 construction. Specifically, the reduction of Blum [Blu83] (instantiated with the Goldreich-Levin hardcore predicate [GL89]) has the following form: $\textsc{Send}(b, (x, r)) = (f(x), r, \langle x, r \rangle \oplus b)$ where x, r are two random strings of length n. Whenever f is one-to-one OWF the resulting function $\textsc{Send}$ is a perfectly binding, non-interactive commitment (see [Gol01a, Construction 4.4.2]). Then, letting $\hat{g}((x, r, b), s)$ be a perfect NC^0 encoding of $\langle x, r \rangle \oplus b$, the function $\widehat{\textsc{Send}}(b, (x, r, s)) = (f(x), r, \hat{g}(x, r, b, s))$ perfectly encodes $\textsc{Send}$, and hence defines a black-box NC^0 reduction from a non-interactive commitment scheme to a one-to-one OWF.

It follows that *non-interactive* commitments in NC^0 are implied by the existence of a 1-1 OWF in $\mathcal{PREN}$ or by the existence of a non-interactive commitment in $\mathcal{PREN}$ (actually, perfect correctness and computational privacy suffice). (We will later show that such a scheme can be based on the intractability of factoring or discrete logarithm. See Remark 5.3.3.)

[21] A modification of this scheme remains secure even if we replace $\textsc{Send}$ with a randomized encoding which is only *statistically*-correct. However, in this modification we cannot use the canonical decommitment stage. Instead, the receiver should verify the decommitment by applying the decoder B to $\hat{c}$ and comparing the result to the computation of the original sender; i.e., the receiver checks whether $B(\hat{c})$ equals to $\textsc{Send}(b, r)$. A disadvantage of this alternative decommitment is that it does not enjoy the enhanced parallelism feature discussed below. Also the resulting scheme is only *statistically* binding.

Interactive commitments

While the existence of an arbitrary OWF is not known to imply non-interactive commitment scheme, it is possible to use OWFs to construct an *interactive* commitment scheme [Nao91]. In particular, the PRG based commitment scheme of [Nao91] has the following simple form: First the receiver chooses a random string $k \in \{0,1\}^{3n}$ and sends it to the sender, then the sender that wishes to commit to the bit b chooses a random string $r \in \{0,1\}^n$ and sends the value of the function $\textsc{Send}(b,k,r)$ to the receiver. (The exact definition of the function $\textsc{Send}(b,k,r)$ is not important in our context.) To decommit the sender sends the randomness r and the bit b and the receiver accepts if $\textsc{Send}(b,k,r)$ equals to the message he had received in the commit phase. Computational hiding requires that for any string family $\{k_n\}$ where $k_n \in \{0,1\}^{3n}$, it holds that $(k_n, \textsc{Send}(0,k_n,U_n)) \overset{c}{\equiv} (k_n, \textsc{Send}(1,k_n,U_n))$. Perfect binding requires that, except with negligible probability (over the randomness of the receiver k), there are no r_0 and r_1 such that $\textsc{Send}(0,k,r_0) = \textsc{Send}(1,k,r_1)$.

Again, if we replace $\textsc{Send}$ by a computationally-private perfectly-correct encoding $\widehat{\textsc{Send}}$, we get a (two-round) interactive commitment scheme (this follows by combining the previous arguments with Fact 2.2.10). Moreover, as in the non-interactive case, when the encoding is in NC^0 the receiver's computation in the decommit phase is in $\mathrm{AND}_n \circ \mathrm{NC}^0$. Since the receiver's computation in the commit phase is also in NC^0, we get an $\mathrm{AND}_n \circ \mathrm{NC}^0$ receiver. (In Remark 5.3.3 we show that such a scheme can be based on the intractability of factoring, discrete logarithm or lattices problems.) As an immediate application, we obtain a constant-round protocol for coin flipping over the phone [Blu83] between an NC^0 circuit and an $\mathrm{AND}_n \circ \mathrm{NC}^0$ circuit.

Statistically hiding commitments

One can apply a similar transformation to other variants of commitment schemes, such as unconditionally hiding (and computationally binding) interactive commitments. (Of course, to preserve the security of such schemes the privacy of the encoding will have to be *statistical* rather than computational.) Unconditionally hiding commitments require some initialization phase, which typically involves a random key sent from the receiver to the sender. We can turn such a scheme into a similar scheme between an NC^0 sender and an $\mathrm{AND}_n \circ \mathrm{NC}^0$ receiver, provided that it conforms to the following structure: (1) the receiver initializes the scheme by *locally* computing a random key k (say, a prime modulus and powers of two group elements for schemes based on discrete logarithm) and sending it to the sender; (2) the sender responds with a single message computed by the commitment function $\textsc{Send}(b,k,r)$ which is in $\mathcal{PREN}$ (actually, perfect correctness and statistical privacy suffice); (3) as in the previous case, the scheme has a canonical decommit stage in which the sender reveals b by sending b and r to the receiver, who verifies that $\textsc{Send}(b,k,r)$ is equal to the commitment c. Statistical hiding requires that for any string family $\{k_n\}$ where $k_n \in \{0,1\}^n$, it holds that $(k_n, \textsc{Send}(0,k_n,U_{m(n)})) \overset{s}{\equiv} (k_n, \textsc{Send}(1,k_n,U_{m(n)}))$, where $m(n)$ is the number of random coins the sender uses. Computational binding requires that, except with negligible probability (over the randomness of the receiver k), an efficient adversary cannot find r_0 and r_1 such that $\textsc{Send}(0,k,r_0) = \textsc{Send}(1,k,r_1)$.

Using the CRHF-based commitment scheme of [DPP94, HM96], one can obtain schemes of the above type based on the intractability of factoring, discrete logarithm, and lattice problems. Given such a scheme, we replace the sender's function by its randomized encoding, and get as a result a statistically hiding commitment scheme whose sender is in NC^0. The new scheme inherits the round complexity of the original scheme and thus consists of only two rounds of interaction. (The security proof is similar to the case of perfectly binding, non-interactive commitment, only this time we use Facts 2.2.5, 2.2.3 instead of Facts 2.2.10, 2.2.8.) If the random key k cannot be computed in $\mathrm{AND}_n \circ \mathrm{NC}^0$ (as in the case of factoring and discrete logarithm based schemes), one can compute k once and for all during the generation of the receiver's circuit and hardwire the key to the receiver's circuit. (See Appendix A.)

Impossibility of an NC^0 receiver

We show that, in any of the above settings, the receiver cannot be realized in NC^0. Recall that in the decommit stage the sender opens his commitment to the bit b by sending a single message (b, m) to the receiver, which accepts or reject it according to b, m and his view v of the commitment stage. (This is the case in all the aforementioned variants.) Suppose that the receiver's computation $f(b, m, v)$ is in NC^0_k. Consider an (honest) execution of the protocol up to the decommit stage in which the sender commits to 1, and the view of the receiver is v. There are two cases: (1) there exists an ambiguous opening m_0, m_1 for which $f(0, m_0) = f(1, m_1) = \mathsf{accept}$; and (2) there is no ambiguous opening, i.e., for all m we have $f(0, m) = \mathsf{reject}$ and $f(1, m_1) = \mathsf{accept}$ for some m_1. We show that we can either break the binding property (in the first case) or the hiding property (in the second case). Indeed, in the first case the sender can choose two random strings m'_0, m'_1 of an appropriate length. The probability that m'_0 (resp., m'_1) agrees with m_0 (resp., m_1) on the bits which are read by the receiver is at least 2^{-k}. Hence, with probability 2^{-2k}, we have $f(0, m'_0) = f(1, m'_1) = \mathsf{accept}$ and the binding property is violated. Now consider case (2). Let r be the substring of m which is read by the receiver. Since $|r| \leq k$ the receiver can efficiently find b (before the decommit stage) by going over all possible r's and checking whether there exists an r such that $f(b, r, v) = \mathsf{accept}$. We conclude that the locality of the receiver's computation f should be super-logarithmic.

4.8.3 Zero-Knowledge Proofs

We move on to the case of zero-knowledge proofs. For simplicity, we begin with the simpler case of non-interactive zero knowledge proofs (NIZK). Such proof systems are similar to standard zero-knowledge protocols except that interaction is traded for the use of a public random string σ to which both the prover and the verifier have a read-only access. More formally, a NIZK (with an efficient prover) for an NP relation $R(x, w)$ is a pair of probabilistic polynomial-time algorithms (P, V) that satisfies the following properties:

- (Completeness) for every $(x, w) \in R$, it holds that $\Pr[V(x, \sigma, P(x, w, \sigma)) = 1] > 1 - \mathrm{neg}(|x|)$;
- (Soundness) for every $x \notin L_R$ (i.e., x such that $\forall w, (x, w) \notin R$) and every prover algorithm P^* we have that $\Pr[V(x, \sigma, P^*(x, \sigma)) = 1] < \mathrm{neg}(|x|)$;
- (Zero-knowledge) there exists a probabilistic polynomial-time simulator M such that for every string sequence $\{(x_n, w_n)\}$ where $(x_n, w_n) \in R$ it holds that $\{(x_n, \sigma, P(x_n, w_n, \sigma))\} \overset{c}{\equiv} \{M(x_n)\}$

(where in all the above σ is uniformly distributed over $\{0, 1\}^{\mathrm{poly}(|x|)}$).

Similarly to the previous cases, we can compile the prover into its computational randomized encoding $\hat{P}$, while the new verifier $\hat{V}$ uses the decoder B to translate the prover's encoded message $\hat{y}$

to the corresponding message of the original prover, and then invokes the original verifier (i.e., $\hat{V} = V(x, \sigma, B(\hat{y}))$). The completeness and soundness of the new protocol follow from the correctness of the encoding. The zero-knowledge property follows from the privacy of the encoding. That is, to simulate the new prover we define a simulator $\hat{M}$ that invokes the simulator M of the original scheme and then applies the simulator S of the encoding to the third entry of M's output. By Fact 2.2.10 and the privacy of $\hat{P}$ it holds that $(x, \sigma, \hat{P}(x, w, \sigma, r)) \overset{c}{\equiv} (x, \sigma, S(P(x, w, \sigma)))$ (where r is the randomness of the encoding $\hat{P}$) while Fact 2.2.8 ensures that $(x, \sigma, S(P(x, w, \sigma))) \overset{c}{\equiv} \hat{M}(x)$.

The above construction generalizes to *interactive* ZK-proofs with an efficient prover. In this case, we can encode the prover's computation (viewed as a function of its input, the NP-witness he holds, his private randomness and all the messages he has received so far), while the new receiver uses the decoder to translate the messages and then invokes the original protocol. The resulting protocol is still computational ZK proof. (The proof is similar to the case of NIZK above, but relies on Fact 2.2.9 instead of Fact 2.2.8.) The same construction works for ZK arguments (in which the soundness holds only against computationally bounded cheating prover). When the encoding is *statistically*-private the above transformation also preserves the statistical ZK property. That is, if the original protocol provides a statistically-close simulation then so does the new protocol.

As before, this general approach does not parallelize the verifier; in fact, the verifier is now required to "work harder" and decode the prover's messages. However, we can improve the verifier's complexity by relying on specific, commitment-based, zero-knowledge protocols from the literature. For instance, in the constant-round protocol for Graph 3-Colorability of [GK96], the computations of the prover and the verifier consist of invoking two commitments (of both types, perfectly binding as well as statistically hiding), in addition to some AC^0 computations. Hence, we can use the parallel commitment schemes described before to construct a constant-round protocol for 3-Colorability between an AC^0 prover and an AC^0 verifier. Since 3-Colorability is NP complete under AC^0-reductions, we get constant-round zero-knowledge proofs in AC^0 for every language in NP. (We will later show that the existence of such a protocol is implied by the intractability of factoring, the discrete logarithm problem, and lattice problems. See Remark 5.3.3.)

Impossibility of an NC^0 verifier. Again, it is impossible to obtain an NC^0 verifier (for non-trivial languages). Suppose that we have a ZK-proof for a language L whose verifier V is in NC^0_k. First, observe that in such a proof system the soundness error is either 0, or larger than $2^{-k} = \Omega(1)$. Hence, since we require a negligible error probability, the soundness must be perfect. Thus, we can efficiently decide whether a string x is in L by letting the verifier interact with the simulator. If $x \in L$ then, except with negligible probability, the verifier should accept (as otherwise we can distinguish between the interaction with the simulator to the interaction with the real prover.) On the other hand, if $x \notin L$ then, due to the perfect soundness, the verifier always rejects. Therefore, $L \in \mathrm{BPP}$. In fact, if the round complexity of the protocol is $t = O(1)$ as in the aforementioned constructions (and the verifier is in NC^0_k), then L must be in NC^0_{tk}. (This is true for any interactive proof protocol, not necessarily ZK.) To see this, note that the verifier computes its output based on at most tk bits of x. If $L \notin \mathrm{NC}^0_{tk}$ then there exist an input $x \in L$ and an input $y \notin L$ which agree on the tk bits read by V. Let v be a view of V which makes him accept x. Then, the same view makes V accept y, in contradiction to the perfect soundness.

4.8.4 Instance hiding schemes

An instance hiding scheme (IHS) allows a powerful machine (an oracle) to help a more limited user compute some function f on the user's input x; the user wishes to keep his input private and so he cannot just send it to the machine. We assume that the user is an algorithm from a low complexity class WEAK whereas the oracle is from a higher complexity class STRONG. In a (non-adaptive, single-oracle) IHS the user first transforms his input x into a (randomized) encrypted instance $y = E(x,r)$ and then asks the oracle to compute $z = g(y)$. The user should be able to recover the value of $f(x)$ from z by applying a decryption algorithm $D(x,r,z)$ (where $D \in \mathsf{WEAK}$) such that $D(x,r,g(E(x,r)) = f(x)$. The hiding of the scheme requires that $E(x,r)$ keeps x secret, i.e., for every string families $\{x_n\}$ and $\{x'_n\}$ (where $|x_n| = |x'_n|$), the ensembles $E(x_n,r)$ and $E(x'_n,r)$ are indistinguishable with respect to functions in STRONG. The default setting of instance hiding considered in the literature refers to a probabilistic polynomial-time user and a computationally unbounded machine. (See [Fei93] for a survey on IHS schemes.) We will scale this down and let the user be an NC^0 function and the oracle be a probabilistic polynomial time machine.

The notion of randomized encoding naturally gives rise to IHS in the following way: Given f we define a related function $h(x,r) = f(x) \oplus r$ (where $|r| = |f(x)|$). Let $\hat{h}((x,r),s)$ be a computational randomized encoding of h whose decoder is B. Then, we define $E(x,(r,s)) = \hat{h}((x,r),s)$, $g(y) = B(y)$ and $D(x,r,z) = r \oplus z$. The correctness of the scheme follows from the correctness of the encoding. To prove the privacy note that, by Fact 2.2.10, it holds that

$$\hat{h}(x_n,r,s) \overset{c}{\equiv} S(f(x_n) \oplus r) \equiv S(f(y_n) \oplus r) \overset{c}{\equiv} \hat{h}(y_n,r,s).$$

Hence, we can construct such a scheme where $\mathsf{WEAK} = \mathrm{NC}^0$ and $\mathsf{STRONG} = \mathcal{CREN}$. (Recall that $\mathcal{CREN} \subseteq \mathrm{BPP}$ and thus computational privacy indeed fools a $\mathcal{CREN}$ oracle.)

4.9 Summary and Discussion

Table 4.9.1 summarizes the properties of randomized encoding that suffice for encoding different cryptographic primitives. (In the case of trapdoor permutations, efficient randomness recovery is also needed.) As mentioned before, in some cases it suffices to use a *computationally-private* randomized encoding. This relaxation allows to construct (some) primitives in NC^0 under more general assumptions. (See Theorem 5.3.1.)

4.9.1 The case of PRFs

It is natural to ask why our machinery cannot be applied to pseudorandom functions (PRFs) (assuming there exists a PRF in $\mathcal{PREN}$), as is implied from the impossibility results of Linial et al. [LMN93]. Suppose that a PRF family $f_k(x) = f(k,x)$ is encoded by the function $\hat{f}(k,x,r)$. There are two natural ways to interpret $\hat{f}$ as a collection: (1) to incorporate the randomness into the key, i.e., $g_{k,r}(x) \overset{\mathrm{def}}{=} \hat{f}(k,x,r)$; (2) to append the randomness to the argument of the collection, i.e., $h_k(x,r) \overset{\mathrm{def}}{=} \hat{f}(k,x,r)$. To rule out the security of approach (1), it suffices to note that the mapping $\hat{f}(\cdot,r)$ is of degree one when r is fixed; thus, to distinguish $g_{k,r}$ from a truly random function, one can check whether the given function is affine (e.g., verify that $g_{k,r}(x) + g_{k,r}(y) = g_{k,r}(x+y) + g_{k,r}(0)$). The same attack applies to the function $h_k(x,r)$ obtained by the second approach, by fixing the randomness r. More generally, the privacy of a randomized encoding is guaranteed only when

Primitive	Encoding	Efficient simulator	Efficient decoder
One-way function	computational	required	—
One-way permutation	perfect	required	—
Trapdoor permutation	perfect	required	required
Pseudorandom generator	perfect	required	—
Collision-resistant hashing	perfect	—	—
Encryption (pub., priv.)	computational	required	required
Signatures, MAC	computational	required	required
Perfectly-binding commitment	perfectly correct comp. private	required	—
Statistically-hiding commitment	perfectly correct stat. private	required	—
Zero-knowledge proof	computational	required	required
Stat. ZK proof/arguments	statistical	required	required
Instance hiding	computational	required	required

Table 4.9.1: Sufficient properties for preserving the security of different primitives.

the randomness is secret and is freshly picked, thus our methodology works well for cryptographic primitives which employ fresh secret randomness in each invocation. PRFs do not fit into this category: while the key contains secret randomness, it is not freshly picked in each invocation.

We finally note that by combining the positive results regarding the existence of various primitives in NC^0 with the fact that PRFs cannot be implemented in NC^0 (as this class is learnable), one can derive a separation between PRFs and other primitives such as PRGs. In particular, there is no NC^0 construction from PRGs to PRFs, unless factoring is easy on the average (more generally, unless there is no PRG in $\oplus\mathrm{L}/poly$). We can also rule out a wide family of AC^0 constructions of PRFs from PRGs (including black-box constructions). Specifically, suppose that there exists an AC^0 construction of PRF F from PRG G which preserves the security of the underlying PRG in the sense that any adversary that breaks F in quasipolynomial time (e.g., $n^{\mathrm{polylog}(n)}$) yields an adversary that breaks G in quasipolynomial time. (This property is respected by black-box reductions.) Then, since any PRF in AC^0 can be broken in quasipolynomial time [LMN93], one can break any PRG in NC^0 in quasipolynomial time, which, by our results, implies a quasipolynomial time algorithm for factoring (on the average), or more generally, a quasipolynomial time attack on every PRG in $\oplus\mathrm{L}/poly$.

4.9.2 Open Problems

The results described in this chapter provide strong evidence for the possibility of cryptography in NC^0. They are also close to optimal in terms of the exact locality that can be achieved. Still, several questions are left for further study. In particular:

- What are the minimal assumptions required for cryptography in NC^0? For instance, does the existence of an arbitrary OWF imply the existence of OWF in NC^0? We show that a OWF in $\mathrm{NL}/poly$ implies a OWF in NC^0.

- Can the existence of a OWF (or PRG) in NC^0_3 be based on general assumptions such are the ones that suffice for implementations in NC^0_4? In Chapters 7 and 8 we construct such a OWF (and even a PRG) under concrete intractability assumptions (e.g., the intractability of decoding a random linear code).
- Can our paradigm for achieving better parallelism be of any practical use?

The above questions motivate a closer study of the complexity of randomized encodings, which so far was only motivated by questions in the domain of secure multiparty computation.

Chapter 5

Computationally Private Randomizing Polynomials and Their Applications

Summary: In this chapter, we study the notion of *computational* randomized encoding (cf. Definition 3.1.7) which relaxes the privacy property of statistical randomized encoding. We construct a computational encoding in NC^0_4 for every *polynomial-time* computable function, assuming the existence of a cryptographic pseudorandom generator (PRG) in $\oplus\mathrm{L}/poly$. (The latter assumption is implied by most standard intractability assumptions used in cryptography.) This result is obtained by combining a variant of Yao's *garbled circuit* technique with previous "information-theoretic" constructions of randomizing polynomials.

We present several applications of computational randomized encoding. In particular, we relax the sufficient assumptions for parallel constructions of cryptographic primitives, obtain new parallel reductions between primitives, and simplify the design of constant-round protocols for multiparty computation.

5.1 Introduction

In Chapter 4 we showed that functions in $\oplus\mathrm{L}/poly$ (resp. NL) admit a perfect (resp. statistical) randomized encoding in NC^0_4. A major question left open by these results is whether every *polynomial-time* computable function admits an encoding in NC^0. In this chapter we consider the relaxed notion of *computational* randomized encoding. As we saw in Chapter 4, computationally private encodings are sufficient for most applications. Thus, settling the latter question for the relaxed notion may be viewed as a second-best alternative.

5.1.1 Overview of Results and Techniques

We construct a computationally private encoding in NC^0_4 for every *polynomial-time* computable function, assuming the existence of a "minimal" cryptographic pseudorandom generator, namely one that stretches its seed by just one bit, in $\oplus\mathrm{L}/poly$.[1] We refer to the latter assumption as the

[1]It is not known whether such a minimal PRG implies a PRG in the same class that stretches its seed by a linear or superlinear amount.

"Easy PRG" (EPRG) assumption. (This assumption can be slightly relaxed, e.g., to also be implied by the existence of a PRG in NL/*poly*; see Remark 5.2.17.) We note that EPRG is a very mild assumption. In particular, it is implied by most concrete intractability assumptions commonly used in cryptography, such as ones related to factoring, discrete logarithm, or lattice problems (see Remark 4.5.8). It is also implied by the existence in $\oplus$L/*poly* of a one-way permutation or, using [HILL99], of any *regular* one-way function (OWF); i.e., a OWF $f = \{f_n\}$ that maps the same (polynomial-time computable) number of elements in $\{0,1\}^n$ to every element in $\mathrm{Im}(f_n)$. (This is the case, for instance, for any one-to-one OWF.)[2] The NC^0 encoding we obtain under the EPRG assumption has degree 3 and locality 4. Its size is nearly linear in the circuit size of the encoded function.

We now give a high-level overview of our construction. Recall that we wish to encode a polynomial-time computable function by an NC^0 function. To do this we rely on a variant of Yao's *garbled circuit* technique [Yao86]. Roughly speaking, Yao's technique allows to efficiently "encrypt" a boolean circuit in a way that enables to compute the output of the circuit but "hides" any other information about the circuit's input. These properties resemble the ones required for randomized encoding.[3] Moreover, the garbled circuit enjoys a certain level of locality (or parallelism) in the sense that gates are encrypted independently of each other. Specifically, each encrypted gate is obtained by applying some cryptographic primitive (typically, a high-stretch PRG or an encryption scheme with special properties), on a constant number of (long) random strings and, possibly, a single input bit. However, the overall circuit might not have constant locality (it might not even be computable in NC) as the cryptographic primitive being used in the gates might be sequential in nature. Thus, the bottleneck of the construction is the parallel time complexity of the primitive being used.

In Chapter 4 we showed (via "information theoretic" randomized encoding) that under relatively mild assumptions many cryptographic primitives can be computed in NC^0. Hence, we can try to plug one of these primitives into the garbled circuit construction in order to obtain an encoding with constant locality. However, a direct use of this approach would require stronger assumptions[4] than EPRG and result in an NC^0 encoding with inferior parameters. Instead, we use the following variant of this approach.

Our construction consists of three steps. The first step is an NC^0 implementation of *one-time symmetric encryption* using a minimal PRG as an oracle. (Such an encryption allows to encrypt a single message whose length may be polynomially larger than the key. Note that we are only assuming the existence of a minimal PRG, i.e., a PRG that stretches its seed only by one bit. Such a PRG cannot be directly used to encrypt long messages.) The second and main step of the construction relies on a variant of Yao's garbled circuit technique [Yao86] to obtain an encoding in NC^0 which uses one-time symmetric encryption as an oracle. By combining these two steps we get an encoding that can be computed by an NC^0 circuit which uses a minimal PRG as an oracle. Finally, using the EPRG assumption and Theorem 4.2.6, we apply a final step of "information-theoretic" encoding to obtain an encoding in NC^0 with degree 3 and locality 4.

[2]Using [HILL99] or [HHR06] this regularity requirement can be relaxed. See Footnote 13 in Chapter 4.

[3]This similarity is not coincidental as both concepts were raised in the context of secure multiparty computation. Indeed, an information theoretic variant of Yao's garbled circuit technique was already used in [IK02] to construct low degree randomized encoding for NC^1 functions.

[4]Previous presentations of Yao's garbled circuit relied on primitives that seem less likely to allow an NC^0 implementation. Specifically, [BMR90, NPS99] require linear stretch PRG and [LP04] requires symmetric encryption that enjoys some additional properties.

The above result gives rise to several types of cryptographic applications, discussed below.

Relaxed assumptions for cryptography in NC^0

In Chapter 4 we showed that the existence of most cryptographic primitives in NC^0 follows from their existence in higher complexity classes such as $\oplus\mathrm{L}/poly$, which is typically a very mild assumption. This result was obtained by combining the results on (information-theoretic) randomized encodings mentioned above with the fact that the security of most cryptographic primitives is inherited by their randomized encoding.

Using our construction of computationally private encodings, we can further relax the sufficient assumptions for cryptographic primitives in NC^0. As we saw in Chapter 4, the security of most primitives is also inherited by their computationally private encoding. This is the case even for relatively "sophisticated" primitives such as public-key encryption, digital signatures, (computationally hiding) commitments, and (interactive or non-interactive) zero-knowledge proofs (see Table 4.9.1). Thus, given that these primitives at all exist,[5] their existence in NC^0 follows from the EPRG assumption, namely from the existence of a PRG in complexity classes such as $\oplus\mathrm{L}/poly$. Previously (using the results of Chapter 4), the existence of each of these primitives in NC^0 would only follow from the assumption that this particular primitive can be implemented in the above classes, a seemingly stronger assumption than EPRG.

It should be noted that we cannot obtain a similar result for some other primitives, such as one-way permutations and collision-resistant hash functions. The results for these primitives obtained in Chapter 4 rely on certain regularity properties of the encoding that are lost in the transition to computational privacy.

Parallel reductions between cryptographic primitives

In Chapter 4 we also obtained new NC^0 *reductions* between cryptographic primitives. (Unlike the results discussed above, here we consider *unconditional* reductions that do not rely on unproven assumptions.) In particular, known NC^1-reductions from PRG to one-way permutations [GL89] or even to more general types of one-way functions [HILL99, Vio05, HHR06] can be encoded into NC^0-reductions (see Remark 4.5.9). However, these NC^0-reductions crucially rely on the very simple structure of the NC^1-reductions from which they are derived. In particular, it is not possible to use the results of Chapter 4 for encoding general NC^1-reductions (let alone polynomial-time reductions) into NC^0-reductions.

As a surprising application of our technique, we get a general "compiler" that converts an arbitrary (polynomial-time) reduction from a primitive $\mathcal{P}$ to a PRG into an NC^0-reduction from $\mathcal{P}$ to a PRG. This applies to all primitives $\mathcal{P}$ that are known to be equivalent to a one-way function, and whose security is inherited by their computationally-private encoding. In particular, we conclude that symmetric encryption,[6] commitment, and digital signatures are all NC^0-reducible to a *minimal* PRG (hence also to a one-way permutation or more general types of one-way functions).

No parallel reductions of this type were previously known, even in NC. The known construction of commitment from a PRG [Nao91] requires a linear-stretch PRG (expanding n bits into $n +$

[5]This condition is redundant in the case of signatures and commitments, whose existence follows from the existence of a PRG. We will later describe a stronger result for such primitives.

[6] By symmetric encryption we refer to (probabilistic) *stateless* encryption for multiple messages, where the parties do not maintain any state information other than the key. If parties are allowed to maintain synchronized states, symmetric encryption can be easily reduced in NC^0 to a PRG.

$\Omega(n)$ bits), which is not known to be reducible *in parallel* to a minimal PRG. Other primitives, such as symmetric encryption and signatures, were not even known to be reducible in parallel to a polynomial-stretch PRG. For instance, the only previous parallel construction of symmetric encryption from a "low-level" primitive is based on the parallel PRF construction of [NR99]. This yields an NC^1-reduction from symmetric encryption to *synthesizers*, a stronger primitive than a PRG. Thus, we obtain better parallelism and at the same time rely on a weaker primitive. The price we pay is that we cannot generally guarantee parallel *decryption*. (See Section 5.3.2 for further discussion.)

An interesting feature of the new reductions is their *non-black-box* use of the underlying PRG. That is, the "code" of the NC^0-reduction we get (implementing $\mathcal{P}$ using an oracle to a PRG) depends on the code of the PRG. This should be contrasted with most known reductions in cryptography, which make a black-box use of the underlying primitive. In particular, this is the case for the abovementioned NC^0-reductions based on Chapter 4. (See [RTV04] for a thorough taxonomy of reductions in cryptography.)

Application to secure computation

The notion of randomizing polynomials was originally motivated by the goal of minimizing the round complexity of secure multi-party computation [Yao86, GMW87, BOGW88, CCD88]. The main relevant observations made in [IK00] were that: (1) the round complexity of most general protocols from the literature is related to the *degree* of the function being computed; and (2) if f is represented by a vector $\hat{f}$ of degree-d randomizing polynomials, then the task of securely computing f can be reduced to that of securely computing some *deterministic* degree-d function $\hat{f}'$ which is closely related to $\hat{f}$. This reduction from f to $\hat{f}'$ is fully *non-interactive*, in the sense that a protocol for f can be obtained by invoking a protocol for $\hat{f}$ and applying a *local* computation on its outputs (without additional interaction).

A useful corollary of our results is that under the EPRG assumption, the task of securely computing an *arbitrary* polynomial-time computable function f reduces (non-interactively) to that of securely computing a related degree-3 function $\hat{f}'$. This reduction is only *computationally* secure. Thus, even if the underlying protocol for $\hat{f}'$ is secure in an information-theoretic sense, the resulting protocol for f will only be computationally secure. (In contrast, previous constructions of randomizing polynomials maintained *information-theoretic* security, but only efficiently applied to restricted function classes such as $\oplus\mathrm{L}/\mathit{poly}$.) This reduction gives rise to new, conceptually simpler, constant-round protocols for general functions. For instance, a combination of our result with the classical "BGW protocol" [BOGW88] gives a simpler, and in some cases more efficient, alternative to the constant-round protocol of Beaver, Micali and Rogaway [BMR90] (though relies on a stronger assumption).

5.1.2 Organization

In Section 5.2 we construct a computationally private encoding in NC^0 for every polynomial-time computable function. Applications of this construction are discussed in Section 5.3. In particular, in Section 5.3.1 we relax the sufficient assumptions for parallel constructions of cryptographic primitives, in Section 5.3.2 we obtain new parallel reductions between primitives, and in Section 5.3.3 simplify the design of constant-round protocols for multiparty computation.

5.2 Computational Encoding in NC^0 for Efficiently Computable Functions

In this section we construct a perfectly correct computational encoding of degree 3 and locality 4 for every efficiently computable function. Our construction consists of three steps. In Section 5.2.1, we describe an NC^0 implementation of one-time symmetric encryption using a *minimal* PRG as an oracle (i.e., a PRG that stretches its seed by just one bit). In Section 5.2.2 we describe the main step of the construction, in which we encode an arbitrary circuit using an NC^0 circuit which uses one-time symmetric encryption as an oracle. This step is based on a variant of Yao's garbled circuit technique [Yao86]. (The privacy proof of this construction is deferred to Section 5.2.4.) Combining the first two steps, we get a computational encoding in NC^0 with an oracle to a minimal PRG. Finally, in Section 5.2.3, we derive the main result by relying on the existence of an "easy PRG", namely, a minimal PRG in $\oplus\mathrm{L}/poly$.

Remark 5.2.1 Recall that the definition of computational randomized encoding uses n both as an input length parameter and as a cryptographic "security parameter" quantifying computational privacy (see Definition 3.1.7). When describing our construction, it will be convenient to use a separate parameter k for the latter, where computational privacy will be guaranteed as long as $k \geq n^\epsilon$ for some constant $\epsilon > 0$.

5.2.1 From PRG to One-Time Encryption

An important tool in our construction is a one-time symmetric encryption; that is, a (probabilistic) private-key encryption that is semantically secure [GM84b] for encrypting a single message. We describe an NC^0-reduction from such an encryption to a minimal PRG, stretching its seed by a single bit. We start by defining minimal PRG and one-time symmetric encryption.

Definition 5.2.2 *Let $G : \{0,1\}^k \to \{0,1\}^{\ell(k)}$ be a PRG (see Definition 4.5.1). We say that G is a* minimal *PRG if it stretches its input by one bit (i.e., $\ell(k) = k+1$). When $\ell(k) = k + \Omega(k)$ we say that G is a* linear-stretch *PRG. We refer to G as a* polynomial-stretch *PRG if $\ell(k) = \Omega(k^c)$ for some constant $c > 1$.*

Definition 5.2.3 (One-time symmetric encryption) *A* one-time symmetric encryption scheme *is a pair (E, D), of probabilistic polynomial-time algorithms satisfying the following conditions:*

- ***Correctness:*** *For every k-bit key e and for every plaintext $m \in \{0,1\}^*$, the algorithms E, D satisfy $D_e(E_e(m)) = m$ (where $E_e(m) \overset{def}{=} E(e,m)$ and similarly for D).*
- ***Security:*** *For every polynomial $\ell(\cdot)$, and every families of plaintexts $\{x_k\}_{k\in\mathbb{N}}$ and $\{x'_k\}_{k\in\mathbb{N}}$ where $x_k, x'_k \in \{0,1\}^{\ell(k)}$, it holds that*

$$\{E_{U_k}(x_k)\}_{k\in\mathbb{N}} \overset{c}{\equiv} \{E_{U_k}(x'_k)\}_{k\in\mathbb{N}}.$$

The integer k serves as the security parameter *of the scheme. The scheme is said to be $\ell(\cdot)$-one-time symmetric encryption scheme if correctness and security hold with respect to plaintexts whose length is bounded by $\ell(k)$.*

The above definition enables to securely encrypt polynomially long messages under short keys. This is an important feature that will be used in our garbled circuit construction described in Section 5.2.2. In fact, it would suffice for our purposes to encrypt messages of some fixed polynomial[7] length, say $\ell(k) = k^2$. This could be easily done in NC^0 if we had oracle access to a PRG with a corresponding stretch. Given such a PRG G, the encryption can be defined by $E_e(m) = G(e) \oplus m$ and the decryption by $D_e(c) = G(e) \oplus c$. However, we would like to base our construction on a PRG with a minimal stretch.

From the traditional "sequential" point of view, such a minimal PRG is equivalent to a PRG with an arbitrary polynomial stretch (cf. [Gol01a, Thm. 3.3.3]). In contrast, this is not known to be the case with respect to parallel reductions. It is not even known whether a linear-stretch PRG is NC-reducible to a minimal PRG (see [Vio05] for some relevant negative results). Thus, a minimal PRG is a more conservative assumption from the point of view of parallel cryptography. Moreover, unlike a PRG with linear stretch, a minimal PRG is reducible in parallel to one-way permutations and other types of one-way functions (see Remark 4.5.9).

The above discussion motivates a *direct* parallel construction of one-time symmetric encryption using a minimal PRG, i.e., a construction that does not rely on a "stronger" type of PRG as an intermediate step. We present such an NC^0 construction below.

Construction 5.2.4 (From PRG to one-time symmetric encryption) *Let G be a minimal PRG that stretches its input by a single bit, let e be a k-bit key, and let m be a $(k+\ell)$-bit plaintext. Define the probabilistic encryption algorithm $E_e(m,(r_1,\ldots,r_{\ell-1})) \overset{def}{=} (G(e) \oplus r_1, G(r_1) \oplus r_2, \ldots, G(r_{\ell-2}) \oplus r_{l-1}, G(r_{\ell-1}) \oplus m)$, where $r_i \leftarrow U_{k+i}$ serve as the coin tosses of E. The decryption algorithm $D_e(c_1,\ldots,c_{\ell-1})$ sets $r_0 = e$, $r_i = c_i \oplus G(r_{i-1})$ for $i = 1,\ldots,\ell$, and outputs r_ℓ.*

We prove the security of Construction 5.2.4 via a standard hybrid argument.

Lemma 5.2.5 *The scheme (E, D) described in Construction 5.2.4 is a one-time symmetric encryption scheme.*

Proof: Construction 5.2.4 can be easily verified to satisfy the correctness requirement. We now prove the security of this scheme. Assume, towards a contradiction, that Construction 5.2.4 is not secure. It follows that there is a polynomial $\ell(\cdot)$ and two families of strings $x = \{x_k\}$ and $y = \{y_k\}$ where $|x_k| = |y_k| = k+\ell(k)$, such that the distribution ensembles $E_e(x_k)$ and $E_e(y_k)$ where $e \leftarrow U_k$, can be distinguished by a polynomial size circuit family $\{A_k\}$ with non-negligible advantage $\varepsilon(k)$.

We use a hybrid argument to derive a contradiction. Fix some k. For a string m of length $k+\ell(k)$ we define for $0 \le i \le \ell(k)$ the distributions $H_i(m)$ in the following way. The distribution $H_0(m)$ is defined to be $E_{r_0}(m,(r_1,\ldots,r_{l-1}))$ where $r_i \leftarrow U_{k+i}$. For $1 \le i \le \ell(k)$, the distribution $H_i(m)$ is defined exactly as $H_{i-1}(m)$ only that the string $G(r_{i-1})$ is replaced with a random string w_{i-1}, which is one bit longer than r_{i-1} (that is, $w_{i-1} \leftarrow U_{k+i}$). Observe that for every $m \in \{0,1\}^{k+\ell(k)}$, all the $\ell(k)$ strings of the hybrid $H_{\ell(k)}(m)$ are distributed uniformly and independently (each of them is the result of XOR with a fresh random string w_i). Therefore, in particular, $H_{\ell(k)}(x_k) \equiv H_{\ell(k)}(y_k)$. Since $H_0(x_k) \equiv E_e(x_k)$ as well as $H_0(y_k) \equiv E_e(y_k)$, it follows that our distinguisher A_k distinguishes, w.l.o.g., between $H_{\ell(k)}(x_k)$ and $H_0(x_k)$ with at least $\varepsilon(k)/2$ advantage. Then, since there are $\ell(k)$ hybrids, there must be $1 \le i \le \ell(k)$ such that the neighboring hybrids, $H_{i-1}(x_k), H_i(x_k)$, can be distinguished by A_k with $\frac{\varepsilon(k)}{2\ell(k)}$ advantage.

[7]Applying the construction to circuits with a bounded fan-out, even linear length would suffice.

We now show how to use A_k to distinguish a randomly chosen string from an output of the pseudorandom generator. Given a string z of length $k+i$ (that is either sampled from $G(U_{k+i-1})$ or from U_{k+i}), we uniformly choose the strings $r_j \in \{0,1\}^{k+j}$ for $j = 1, \ldots, \ell(k)-1$. We feed A_k with the sample $(r_1, \ldots, r_{i-1}, z \oplus r_i, G(r_i) \oplus r_{i+1}, \ldots, G(r_{\ell(k)-1}) \oplus x_k)$. If z is a uniformly chosen string then the above distribution is equivalent to $H_i(x_k)$. On the other hand, if z is drawn from $G(U_i)$ then the result is distributed exactly as $H_{i-1}(x_k)$, since each of the first $i-1$ entries of $H_{i-1}(x_k)$ is distributed uniformly and independently of the remaining entries (each of these entries was XOR-ed with a fresh and unique random w_j). Hence, we constructed an adversary that breaks the PRG with non-negligible advantage $\frac{\varepsilon(k)}{2\ell(k)}$, deriving a contradiction. ■

Since the encryption algorithm described in Construction 5.2.4 is indeed an NC^0 circuit with oracle access to a minimal PRG, we get the following lemma.

Lemma 5.2.6 *Let G be a minimal PRG. Then, there exists one-time symmetric encryption scheme (E, D) in which the encryption function E is in $\mathrm{NC}^0[G]$.*

Note that the decryption algorithm of the above construction is sequential. We can parallelize it (without harming the parallelization of the encryption) at the expense of strengthening the assumption we use.

Claim 5.2.7 *Let PG (resp. LG) be a polynomial-stretch (resp. linear-stretch) PRG. Then, for every polynomial $p(\cdot)$ there exists a $p(\cdot)$-one-time symmetric encryption scheme (E, D) such that $E \in \mathrm{NC}^0[PG]$ and $D \in \mathrm{NC}^0[PG]$ (resp. $E \in \mathrm{NC}^0[LG]$ and $D \in \mathrm{NC}^1[LG]$).*

Proof: Use Construction 5.2.4 (where $|r_i| = |G(r_{i-1})|$). When the stretch of G is polynomial (resp. linear) the construction requires only $O(1)$ (resp. $O(\log k)$) invocations of G, and therefore, so does the decryption algorithm. ■

5.2.2 From One-Time Encryption to Computational Encoding

Let $f = \{f_n : \{0,1\}^n \to \{0,1\}^{\ell(n)}\}_{n \in \mathbb{N}}$ be a polynomial-time computable function, computed by the uniform circuit family $\{C_n\}_{n \in \mathbb{N}}$. We use a one-time symmetric encryption scheme (E, D) as a black box to encode f by a perfectly correct computational encoding $\hat{f} = \{\hat{f}_n\}_{n \in \mathbb{N}}$. Each $\hat{f}_n$ will be an NC^0 circuit with an oracle access to the encryption algorithm E, where the latter is viewed as a function of the key, the message, and its random coin tosses. The construction uses a variant of Yao's garbled circuit technique [Yao86]. Our notation and terminology for this section borrow from previous presentations of Yao's construction in [Rog91, NPS99, LP04].[8] Before we describe the actual encoding it will be convenient to think of the following "physical" analog that uses locks and boxes.

[8]Security proofs for variants of this construction were given implicitly in [Rog91, TX03, LP04] in the context of secure computation. However, they cannot be directly used in our context for different reasons. In particular, the analysis of [LP04] relies on a special form of symmetric encryption and does not achieve perfect correctness, while that of [Rog91, TX03] relies on a linear-stretch PRG.

A physical encoding. To each wire of the circuit we assign a pair of keys: a 0-key that represents the value 0 and a 1-key that represents the value 1. For each of these pairs we randomly color one key black and the other key white. This way, given a key one cannot tell which bit it represents (since the coloring is random). For every gate of the circuit, the encoding consists of four double-locked boxes – a white-white box (which is locked by the white keys of the wires that enter the gate), a white-black box (locked by the white key of the left incoming wire and the black key of the right incoming wire), a black-white box (locked by the black key of the left incoming wire and the white key of the right incoming wire) and a black-black box (locked by the black keys of the incoming wires). Inside each box we put one of the keys of the gate's output wires. Specifically, if a box is locked by the keys that represent the values α, β then for every outgoing wire we put in the box the key that represents the bit $g(\alpha, \beta)$, where g is the function that the gate computes. For example, if the gate is an OR gate then the box which is locked by the incoming keys that represent the bits $(0, 1)$ contains all the 1-keys of the outgoing wires. So if one has a single key for each of the incoming wires, he can open only one box and get a single key for each of the outgoing wires. Moreover, as noted before, holding these keys does not reveal any information about the bits they represent.

Now, fix some input x for f_n. For each wire, exactly one of the keys corresponds to the value of the wire (induced by x); we refer to this key as the *active key* and to the second key as the *inactive* key. We include in the encoding of $f_n(x)$ the active keys of the input wires. (This is the only place in which the encoding depends on the input x.) Using these keys and the locked boxes as described above, one can obtain the active keys of all the wires by opening the corresponding boxes in a bottom-to-top order. To make this information useful (i.e., to enable decoding of $f_n(x)$), we append to the encoding the semantics of the output wires; namely, for each output wire we expose whether the 1-key is white or black. Hence, the knowledge of the active key of an output wire reveals the value of the wire.

The actual encoding. The actual encoding is analogous to the above physical encoding. We let random strings play the role of physical keys. Instead of locking a value in a double-locked box, we encrypt it under the XOR of two keys. Before formally defining the construction, we need the following notation. Denote by $x = (x_1, \ldots, x_n)$ the input for f_n. Let $k = k(n)$ be a security parameter which may be set to n^{ε} for an arbitrary positive constant ε (see Remark 5.2.1). Let $\Gamma(n)$ denote the number of gates in C_n. For every $1 \leq i \leq |C_n|$, denote by $b_i(x)$ the value of the i-th wire induced by the input x; when x is clear from the context we simply use b_i to denote the wire's value.

Our encoding $\hat{f}_n(x, (r, W))$ consists of random inputs of two types: $|C_n|$ pairs of strings $W_i^0, W_i^1 \in \{0, 1\}^{2k}$, and $|C_n|$ bits (referred to as masks) denoted $r_1, \ldots, r_{|C_n|}$.[9] The strings W_i^0, W_i^1 will serve as the 0-key and the 1-key of the i-th wire, while the bit r_i will determine which of these keys is the black key. We use c_i to denote the value of wire i masked by r_i; namely, $c_i = b_i \oplus r_i$. Thus, c_i is the color of the active key of the i-th wire (with respect to the input x). As before, the encoding $\hat{f}_n(x, (r, W))$ will reveal each active key $W_i^{b_i}$ and its color c_i but will hide the inactive keys $W_i^{1-b_i}$ and the masks r_i of all the wires (except the masks of the output wires). Intuitively, since the active keys and inactive keys are distributed identically, the knowledge of an active key $W_i^{b_i}$ does not reveal the value b_i.

[9]In fact, each application of the encryption scheme will use some additional random bits. To simplify notation, we keep these random inputs implicit.

The encoding $\hat{f}_n$ consists of the concatenation of $O(|C_n|)$ functions, which include several entries for each gate and for each input and output wire. In what follows $\oplus$ denotes bitwise-xor on strings; when we want to emphasize that the operation is applied to single bits we will usually denote it by either $+$ or $-$. We use $\circ$ to denote concatenation. For every $\beta \in \{0,1\}$ and every i, we view the string W_i^{β} as if it is partitioned into two equal-size parts denoted $W_i^{\beta,0}, W_i^{\beta,1}$.

Construction 5.2.8 *Let C_n be a circuit that computes f_n. Then, we define $\hat{f}_n(x,(r,W))$ to be the concatenation of the following functions of $(x,(r,W))$.*

Input wires: *For an input wire i, labeled by a literal ℓ (either some variable x_u or its negation) we append the function $W_i^{\ell} \circ (\ell + r_i)$.*

Gates: *Let $t \in [\Gamma(n)]$ be a gate that computes the function $g \in \{\mathrm{AND}, \mathrm{OR}\}$ with input wires i, j and output wires $y_1, \ldots, y_m$. We associate with this gate 4 functions that are referred to as gate labels. Specifically, for each of the 4 choices of $a_i, a_j \in \{0,1\}$, we define a corresponding function $Q_t^{a_i,a_j}$. This function can be thought of as the box whose color is (a_i, a_j). It is defined as follows:*

$$Q_t^{a_i,a_j}(r,W) \overset{def}{=} E_{W_i^{a_i - r_i, a_j} \oplus W_j^{a_j - r_j, a_i}}\Big(W_{y_1}^{g(a_i - r_i, a_j - r_j)} \circ (g(a_i - r_i, a_j - r_j) + r_{y_1}) \circ \ldots \circ W_{y_m}^{g(a_i - r_i, a_j - r_j)} \circ (g(a_i - r_i, a_j - r_j) + r_{y_m})\Big), \tag{5.2.1}$$

where E is a one-time symmetric encryption algorithm. (For simplicity, the randomness of E is omitted.) That is, the colored keys of all the output wires of this gate are encrypted under a key that depends on the keys of the input wires of the gate. Note that $Q_t^{a_i,a_j}$ depends only on the random inputs. We refer to the label $Q_t^{c_i,c_j}$ that is indexed by the colors of the active keys of the input wires as an active label, *and to the other three labels as the* inactive labels.

Output wires: *For each output wire i of the circuit, we add the mask of this wire r_i.*

It is not hard to verify that $\hat{f}_n$ is in $\mathrm{NC}^0[E]$. In particular, a term of the form W_i^{ℓ} is a 3-local function of W_i^0, W_i^1 and ℓ, since its j-th bit depends on the j-th bit of W_i^0, the j-th bit of W_i^1 and on the literal ℓ. Similarly, the keys that are used in the encryptions are 8-local functions, and the arguments to the encryption are 6-local functions of (r,W).

We will now analyze the complexity of $\hat{f}_n$. The output complexity and randomness complexity of $\hat{f}$ are both dominated by the complexity of the gate labels. Generally, the complexity of these functions is $\mathrm{poly}(|C_n| \cdot k)$ (since the encryption E is computable in polynomial time).[10] However, when the circuit C_n has bounded fan-out (say 2) each invocation of the encryption uses $\mathrm{poly}(k)$ random bits and outputs $\mathrm{poly}(k)$ bits. Hence, the overall complexity is $O(|C_n|) \cdot \mathrm{poly}(k) = O(|C_n| \cdot n^{\varepsilon})$ for an arbitrary constant $\varepsilon > 0$. Since any circuit with unbounded fan-out of size $|C_n|$ can be (efficiently) transformed into a bounded-fanout circuit whose size is $O(|C_n|)$ (at the price of a logarithmic factor in the depth), we get an encoding of size $O(|C_n| \cdot n^{\varepsilon})$ for every (unbounded fan-out) circuit family $\{C_n\}$.

Let $\mu(n), s(n)$ be the randomness complexity and the output complexity of $\hat{f}_n$ respectively. We claim that the function family $\hat{f} = \{\hat{f}_n : \{0,1\}^n \times \{0,1\}^{\mu(n)} \to \{0,1\}^{s(n)}\}_{n \in \mathbb{N}}$ defined above is indeed a computationally randomized encoding of the family f. We start with perfect correctness.

[10]Specifically, the encryption is always invoked on messages whose length is bounded by $\ell(n) \overset{\mathrm{def}}{=} O(|C_n| \cdot k)$, hence we can use $\ell(n)$-one-time symmetric encryption.

Lemma 5.2.9 (Perfect correctness) *There exists a polynomial-time decoder algorithm B such that for every $n \in \mathbb{N}$ and every $x \in \{0,1\}^n$ and $(r, W) \in \{0,1\}^{\mu(n)}$, it holds that*

$$B(1^n, \hat{f}_n(x, (r, W))) = f_n(x).$$

Proof: Let $\alpha = \hat{f}_n(x, (r, W))$ for some $x \in \{0,1\}^n$ and $(r, W) \in \{0,1\}^{\mu(n)}$. Given α, our decoder computes, for every wire i, the active key $W_i^{b_i}$ and its color c_i. Then, for an output wire i, the decoder retrieves the mask r_i from α and computes the corresponding output bit of $f_n(x)$; i.e., outputs $b_i = c_i - r_i$. (Recall that the masks of the output wires are given explicitly as part of α.) The active keys and their colors are computed by scanning the circuit from bottom to top.

For an input wire i the desired value, $W_i^{b_i} \circ c_i$, is given as part of α. Next, consider a wire y that goes out of a gate t, and assume that we have already computed the desired values of the input wires i, j of this gate. We use the colors c_i, c_j of the active keys of the input wires to select the active label $Q_t^{c_i,c_j}$ of the gate t (and ignore the other 3 inactive labels of this gate). Consider this label as in Equation (5.2.1); recall that this cipher was encrypted under the key $W_i^{c_i - r_i, c_j} \oplus W_j^{c_j - r_j, c_i}$ $= W_i^{b_i,c_j} \oplus W_j^{b_j,c_i}$. Since we have already computed the values $c_i, c_j, W_i^{b_i}$ and $W_j^{b_j}$, we can decrypt the label $Q_t^{c_i,c_j}$ (by applying the decryption algorithm D). Hence, we can recover the encrypted plaintext, that includes, in particular, the value $W_y^{g(b_i,b_j)} \circ (g(b_i, b_j) + r_y)$, where g is the function that gate t computes. Since by definition $b_y = g(b_i, b_j)$, the decrypted string contains the desired value. ■

Remark 5.2.10 By the description of the decoder it follows that if the circuit C_n is in NC^i, then the decoder is in NC$^i[D]$, where D is the decryption algorithm. In particular if D is in NC^j then the decoder is in NC^{i+j}. This fact will be useful for some of the applications discussed in Section 5.3.

To argue computational privacy we need to prove the following lemma, whose proof is deferred to Section 5.2.4.

Lemma 5.2.11 (Computational privacy) *There exists a probabilistic polynomial-time simulator S, such that for any family of strings $\{x_n\}_{n \in \mathbb{N}}$, $|x_n| = n$, it holds that $S(1^n, f_n(x_n)) \overset{c}{\equiv} \hat{f}_n(x_n, U_{\mu(n)})$.*

Remark 5.2.12 (Information-theoretic variant) Construction 5.2.8 can be instantiated with a *perfect* (information-theoretic) encryption scheme, yielding a perfectly private randomized encoding. (The privacy proof given in Section 5.2.4 can be easily modified to treat this case.) However, in such an encryption the key must be as long as the encrypted message [Sha49]. It follows that the wires' key length grows exponentially with their distance from the outputs, rendering the construction efficient only for NC^1 circuits. This information-theoretic variant of the garbled circuit construction was previously suggested in [IK02]. We will use it in Section 5.2.3 for obtaining a computational encoding with a parallel decoder.

5.2.3 Main Results

Combining Lemmas 5.2.9, 5.2.11, and 5.2.6 we get an NC^0 encoding of any efficiently computable function using an oracle to a minimal PRG.

Theorem 5.2.13 *Suppose f is computed by a uniform family $\{C_n\}$ of polynomial-size circuits. Let G be a (minimal) PRG. Then, f admits a perfectly correct computational encoding $\hat{f}$ in* $\mathrm{NC}^0[G]$. *The complexity of $\hat{f}$ is $O(|C_n| \cdot n^{\varepsilon})$ (for an arbitrary constant $\varepsilon > 0$).*

We turn to the question of eliminating the PRG oracles. We follow the natural approach of replacing each oracle with an NC^0 implementation. (A more general but less direct approach will be described in Remark 5.2.17.) Using Theorem 4.5.6, a minimal PRG in NC^0 is implied by a PRG in $\mathcal{PREN}$, and in particular by a PRG in NC^1 or even $\oplus\mathrm{L}/poly$. Thus, we can base our main theorem on the following "easy PRG" assumption.

Assumption 5.2.14 (Easy PRG (EPRG)) *There exists a PRG in* $\oplus\mathrm{L}/poly$.

As discussed in Section 5.1.1, EPRG is a very mild assumption. In particular, it is implied by most standard cryptographic intractability assumptions, and is also implied by the existence in $\oplus\mathrm{L}/poly$ of one-way permutations and other types of one-way functions.

Combining Theorem 5.2.13 with the EPRG assumption, we get a computational encoding in NC^0 for every efficiently computable function. To optimize its parameters we apply a final step of perfect encoding, yielding a computational encoding with degree 3 and locality 4 (see Corollary 4.2.9). Thus, we get the following main theorem.

Theorem 5.2.15 *Suppose f is computed by a uniform family $\{C_n\}$ of polynomial-size circuits. Then, under the EPRG assumption, f admits a perfectly correct computational encoding $\hat{f}$ of degree 3, locality 4 and complexity $O(|C_n| \cdot n^{\varepsilon})$ (for an arbitrary constant $\varepsilon > 0$).*

Corollary 5.2.16 *Under the EPRG assumption, $\mathcal{CREN} = \mathrm{BPP}$.*

Proof: Let $f(x)$ be a function in BPP. It follows that there exists a function $f'(x, z) \in \mathrm{P}$ such that for every $x \in \{0,1\}^n$ it holds that $\Pr_z[f'(x, z) \neq f(x)] \leq 2^{-n}$. Let $\hat{f}'((x, z), r)$ be the NC^0 computational encoding of f' promised by Theorem 5.2.15. Since f' is a statistical encoding of f (the simulator and the decoder are simply the identity functions), it follows from Lemma 3.2.4 that $\hat{f}(x, (z, r)) \overset{\mathrm{def}}{=} \hat{f}'((x, z), r)$ is a computational encoding of f in NC^0.

Conversely, suppose $f \in \mathcal{CREN}$ and let $\hat{f}$ be an NC^0 computational encoding of f. A BPP algorithm for f can be obtained by first computing $\hat{y} = \hat{f}(x, r)$ on a random r and then invoking the decoder on $\hat{y}$ to obtain the output $y = f(x)$ with high probability. ■

Remark 5.2.17 (Relaxing the EPRG assumption) The EPRG assumption is equivalent to the existence of a PRG in NC^0 or in $\mathcal{PREN}$. It is possible to base Theorem 5.2.15 on a seemingly more liberal assumption by taking an alternative approach that does not rely on a *perfect* encoding. The idea is to first replace each PRG oracle with an implementation G from some class $\mathcal{C}$, and only then apply a (perfectly correct) *statistical* encoding to the resulting $\mathrm{NC}^0[G]$ circuit. Thus, we need G to be taken from a class $\mathcal{C}$ such that $\mathrm{NC}^0[\mathcal{C}] \subseteq \mathcal{SREN}$. It turns out that the class $\mathrm{NL}/poly$ satisfies this property. In particular, by [Imm88] it holds that $\mathrm{NC}^0[\mathrm{NL}/poly] = \mathrm{NL}/poly$, and by Theorem 4.2.8 we have $\mathrm{NL}/poly \subseteq \mathcal{SREN}$ (and furthermore, functions in $\mathrm{NL}/poly$ admit a statistical NC^0 encoding with perfect correctness). Thus, we can replace $\oplus\mathrm{L}/poly$ in the EPRG assumption with $\mathrm{NL}/poly$. Another alternative is to assume the existence of a one-time symmetric encryption (E, D) whose encrypting algorithm E is in $\mathcal{SREN}$. According to Theorem 4.7.3 the last assumption is equivalent to the existence of one-time symmetric encryption (with negligible decryption error) whose encryption algorithm is in NC^0. Hence by plugging this scheme to Construction 5.2.8, we obtain a computationally private, *statistically* correct randomized encoding in NC^0 for any polynomial-time computable function (and in particular derive Corollary 5.2.16). In fact, we can even obtain perfect correctness (as in Theorem 5.2.15) by assuming the existence of one-time symmetric encryption in $\mathrm{NL}/poly$ (which, using Theorems 4.7.3, 4.2.8, implies such an errorless scheme in NC^0). Note that (PRG $\in \oplus\mathrm{L}/poly$) $\implies$ (PRG $\in \mathrm{NL}/poly$) $\implies$ (one-time encryption $\in \mathrm{NL}/poly$), while the converse implications are not known to hold.

On the parallel complexity of the decoder. As we shall see in Section 5.3, it is sometimes useful to obtain a computational encoding whose decoder is also parallelized. Recall that if the circuit computing f is an NC^i circuit and the decryption algorithm (used in the construction) is in NC^j, we obtain a parallel decoder in NC^{i+j} (see Remark 5.2.10). Unfortunately, we cannot use the parallel symmetric encryption scheme of Construction 5.2.4 for this purpose because of its sequential decryption.

We can get around this problem by strengthening the EPRG assumption. Suppose we have a *polynomial-stretch* PRG in NC^1. (This is implied by some standard cryptographic assumptions, see [NR04].) In such a case, by Claim 5.2.7, we can obtain a one-time symmetric encryption scheme (E, D) (for messages of a fixed polynomial length) in which both E and D are in NC^1. Our goal is to turn this into a scheme $(\hat{E}, \hat{D})$ in which the encryption $\hat{E}$ is in NC^0 and the decryption is still in NC^1. We achieve this by applying to (E, D) the encoding given by the information-theoretic variant of the garbled circuit construction (see Remark 5.2.12 or [IK02]). That is, $\hat{E}$ is a (perfectly correct and private) NC^0 encoding of E, and $\hat{D}$ is obtained by composing D with the decoder of the information-theoretic garbled circuit. (The resulting scheme $(\hat{E}, \hat{D})$ is still a secure encryption scheme, see Theorem 4.7.3.) Since the symmetric encryption (E', D') employed by the information-theoretic garbled circuit is in NC^0, its decoder can be implemented in $\mathrm{NC}^1[D'] = \mathrm{NC}^1$. Thus, $\hat{D}$ is also in NC^1 (as $\mathrm{NC}^0[\text{decoder}] = \mathrm{NC}^1$). Combining this encryption scheme with Construction 5.2.8, we get a computational encoding of a function $f \in \mathrm{NC}^i$ with encoding in NC^0 and decoding in NC^{i+1}. Assuming there exists a linear-stretch PRG in NC^1, we can use a similar argument to obtain an NC^0 encoding for f whose decoding in NC^{i+2}. (In this case we use the linear-PRG part of Claim 5.2.7.) Summarizing, we have the following:

Claim 5.2.18 *Suppose there exists a PRG with polynomial stretch (resp. linear stretch) in* NC^1. *Then, every function* $f \in \mathrm{NC}^i$ *admits a perfectly-correct computational encoding in* NC^0 *whose decoder is in* NC^{i+1} *(resp.* NC^{i+2}*).*

5.2.4 Proof of Lemma 5.2.11

The simulator

We start with the description of the simulator S. Given 1^n and $f_n(x)$, for some $x \in \{0,1\}^n$, the simulator chooses, for every wire i of the circuit C_n, an active key and a color; namely, S selects a random string $W_i^{b_i}$ of length $2k(n)$, and a random bit c_i. (Recall that b_i denotes the value of the i-th wire induced by the input x. The simulator, of course, does not know this value.) For an input wire i, the simulator outputs $W_i^{b_i} \circ c_i$. For a gate t with input wires i, j and output wires $y_1, \ldots, y_m$ the simulator computes the active label $Q_t^{c_i,c_j} = E_{W_i^{b_i,c_j} \oplus W_j^{b_j,c_i}}(W_{y_1}^{b_{y_1}} \circ c_{y_1} \circ \ldots \circ W_{y_m}^{b_{y_m}} \circ c_{y_m})$ and sets the other three inactive labels of this gate to be encryptions of all-zeros strings of appropriate length under random keys; that is, for every two bits $(a_i, a_j) \neq (c_i, c_j)$, the simulator chooses uniformly a $k(n)$-bit string R_{a_i,a_j} and outputs $Q_l^{a_i,a_j} = E_{R_{a_i,a_j}}(0^{|W_{y_1}^{b_{y_1}} \circ c_{y_1} \circ \ldots \circ W_{y_m}^{b_{y_m}} \circ c_{y_m}|})$. Finally, for an output wire i, the simulator outputs $r_i = c_i - b_i$ (recall that b_i is known since $f_n(x)$ is given).

Since C_n can be constructed in polynomial time and since the encryption algorithm runs in polynomial time the simulator is also a polynomial-time algorithm. We refer to the gate labels constructed by the simulator as "fake" gate labels and to gate labels of $\hat{f}_n$ as "real" gate labels.

Assume, towards a contradiction, that there exists a (non-uniform) polynomial-size circuit family $\{A_n\}$, a polynomial $p(\cdot)$, a string family $\{x_n\}$, $|x_n| = n$, such that for infinitely many n's it holds that

$$\Delta(n) \stackrel{\text{def}}{=} |\Pr[A_n(S(1^n, f_n(x_n))) = 1] - \Pr[A_n(\hat{f}_n(x_n, U_{\mu(n)})) = 1]| > \frac{1}{p(n)}.$$

We use a hybrid argument to show that such a distinguisher can be used to break the encryption hence deriving a contradiction.

Hybrid encodings

From now on we fix n and let $k = k(n)$. We construct a sequence of hybrid distributions that depend on x_n, and mix "real" gates labels and "fake" ones, such that one endpoint corresponds to the simulated output (in which all the gates have "fake" labels) and the other endpoint corresponds to $\hat{f}_n(x_n, U_{\mu(n)})$ (in which all the gates have real labels). Hence, if the extreme hybrids can be efficiently distinguished then there must be two neighboring hybrids that can be efficiently distinguished.

The hybrids H_t^n. First, we order the gates of C_n in topological order. That is, if the gate t uses the output of gate t', then $t' < t$. Now, for every $t = 0, \ldots, \Gamma(n)$, we define the hybrid algorithm H_t^n that constructs "fake" labels for the first t gates and "real" labels for the rest of the gates:

1. For every wire i uniformly choose two $2k$-bit strings $W_i^{b_i}, W_i^{1-b_i}$ and a random bit c_i.

2. For every input wire i output $W_i^{b_i} \circ c_i$.

3. For every gate $t' \leq t$ with input wires i, j and output wires $y_1, \ldots, y_m$ output

$$Q_{t'}^{c_i,c_j} = E_{W_i^{b_i,c_j} \oplus W_j^{b_j,c_i}}(W_{y_1}^{b_{y_1}} \circ c_{y_1} \circ \ldots \circ W_{y_m}^{b_{y_m}} \circ c_{y_m}),$$

   and for every choice of $(a_i, a_j) \in \{0,1\}^2$ that is different from (c_i, c_j), uniformly choose a k-bit string R_{a_i,a_j} and output $Q_{t'}^{a_i,a_j} = E_{R_{a_i,a_j}}(0^{|W_{y_1}^{b_{y_1}} \circ c_{y_1} \circ \ldots \circ W_{y_m}^{b_{y_m}} \circ c_{y_m}|})$.

4. For every gate $t' > t$, let g be the function that t' computes (AND or OR), let i, j be the input wires of t' and let $y_1, \ldots, y_m$ be its output wires. Use x_n to compute the value of $b_i(x_n), b_j(x_n)$, and set $r_i = c_i - b_i$ and $r_j = c_j - b_j$. For every choice of $(a_i, a_j) \in \{0,1\}^2$, compute $Q_{t'}^{a_i,a_j}$ exactly as in Equation 5.2.1, and output it.

5. For every output wire i compute b_i and output $r_i = c_i - b_i$.

Claim 5.2.19 *There exist some* $0 \leq t \leq \Gamma(n)-1$ *such that* A_n *distinguishes between* H_t^n *and* H_{t+1}^n *with advantage* $\frac{\Delta(n)}{\Gamma(n)}$.

Proof: First, note that H_t^n uses the string x_n only when constructing real labels, that is in Step 4. Steps 1–3 can be performed without any knowledge on x_n, and Step 5 requires only the knowledge of $f_n(x_n)$. Obviously, the algorithm $H_{\Gamma(n)}^n$ is just a different description of the simulator S, and therefore $S(1^n, f_n(x_n)) \equiv H_{\Gamma(n)}^n$. We also claim that the second extreme hybrid, H_0^n coincides with the distribution of the "real" encoding, $\hat{f}_n(x_n, U_{\mu(n)})$. To see this note that (1) the strings W_i^0, W_i^1 are chosen uniformly and independently by H_0^n, as they are in $\hat{f}_n(x_n, U_{\mu(n)})$; and (2) since H_0^n chooses the c_i's uniformly and independently and sets $r_i = c_i - b_i$ then the r_i's themselves are also distributed uniformly and independently exactly as they are in $\hat{f}_n(x_n, U_{\mu(n)})$. Since for every gate t the value of $Q_t^{a_i,a_j}$ is a function of the random variables, and since it is computed by H_0^n in the same way as in $\hat{f}_n(x_n, U_{\mu(n)})$, we get that $H_0^n \equiv \hat{f}_n(x_n, U_{\mu(n)})$.

Hence, we can write

$$\Delta(n) = |\Pr[A_n(H_0^n) = 1] - \Pr[A_n(H_{\Gamma(n)}^n) = 1]| \leq \sum_{t=0}^{\Gamma(n)-1} |\Pr[A_n(H_t^n) = 1] - \Pr[A_n(H_{t+1}^n) = 1]|,$$

and so there exists some $0 \leq t \leq \Gamma(n) - 1$ such that A_n distinguishes between H_t^n and H_{t+1}^n with advantage $\frac{\Delta(n)}{\Gamma(n)}$. ■

Distinguishing fake gates from real gates

We now show that distinguishing H_t^n and H_{t+1}^n allows to distinguish whether a single gate is real or fake. To do this, we define two random experiments $P_n(0)$ and $P_n(1)$, that produce a real gate and a fake gate, correspondingly.

The experiments $P_n(0), P_n(1)$. Let i, j be the input wires of the gate t, let $y_1, \ldots, y_m$ be the output wires of t, let g be the function that gate t computes, and let $b_i, b_j, b_{y_1}, \ldots, b_{y_m}$ be the values of the corresponding wires induced by the input x_n. For $\sigma \in \{0, 1\}$, define the distribution $P_n(\sigma)$ as the output distribution of the following random process:

- Uniformly choose the $2k$-bit strings $W_i^{b_i}, W_i^{1-b_i}, W_j^{b_j}, W_j^{1-b_j}, W_{y_1}^{b_{y_1}}, W_{y_1}^{1-b_{y_1}}, \ldots, W_{y_m}^{b_{y_m}}, W_{y_m}^{1-b_{y_m}}$, and the random bits $c_i, c_j, c_{y_1}, \ldots, c_{y_m}$.
- If $\sigma = 0$ then set $Q_t^{c_i,c_j}$ and the other three $Q_t^{a_i,a_j}$ exactly as in Step 3 of H_t^n.
- If $\sigma = 1$ then set $Q_t^{a_i,a_j}$ exactly as in Step 4 of H_t^n; that is, set $r_i = c_i - b_i$, $r_j = c_j - b_j$, and for every choice of $(a_i, a_j) \in \{0, 1\}^2$, let

$$Q_t^{a_i,a_j} = E_{W_i^{a_i-r_i,a_j} \oplus W_j^{a_j-r_j,a_i}} \Big(W_{y_1}^{g(a_i-r_i,a_j-r_j)} \circ (g(a_i - r_i, a_j - r_j) + r_{y_1}) \circ \ldots \circ W_{y_m}^{g(a_i-r_i,a_j-r_j)} \circ (g(a_i - r_i, a_j - r_j) + r_{y_m}) \Big).$$

- Output $(W_i^{b_i}, W_j^{b_j}, W_{y_1}^{b_{y_1}}, W_{y_1}^{1-b_{y_1}}, \ldots, W_{y_m}^{b_{y_m}}, W_{y_m}^{1-b_{y_m}}, c_i, c_j, c_{y_1}, \ldots, c_{y_m}, Q_t^{0,0}, Q_t^{0,1}, Q_t^{1,0}, Q_t^{1,1})$.

Claim 5.2.20 *There exist a polynomial size circuit A'_n that distinguishes between $P_n(0)$ and $P_n(1)$ with advantage $\frac{\Delta(n)}{\Gamma(n)}$.*

Proof: The adversary A'_n uses the output of $P_n(\sigma)$ to construct one of the hybrids H^n_t and H^n_{t+1}, and then uses A_n to distinguish between them. Namely, given the output of P_n, the distinguisher A'_n invokes the algorithm H^n_t where the values of $(W_i^{b_i}, W_j^{b_j}, W_{y_1}^{b_{y_1}}, W_{y_1}^{1-b_{y_1}}, \ldots, W_{y_m}^{b_{y_m}}, W_{y_m}^{1-b_{y_m}}, c_i, c_j, c_{y_1}, \ldots, c_{y_m}, Q_t^{0,0}, Q_t^{0,1}, Q_t^{1,0}, Q_t^{1,1})$ are set to the values given by P_n. By the definition of P_n, when $P_n(0)$ is invoked we get the distribution of H^n_t, that is the gate t is "fake"; on the other hand, if $P_n(1)$ is invoked then the gate t is "real" and we get the distribution of H^n_{t+1}. Hence, by Claim 5.2.19, A'_n has the desired advantage. Finally, since H^n_t runs in polynomial time (when x_n is given), the size of A'_n is indeed polynomial. ■

A delicate point. Note that P_n does not output the inactive keys of the wires i and j (which is crucial for Claim 5.2.21 to hold). However, the hybrid distributions use inactive keys of wires that either enter a real gate or leave a real gate (in the first case the inactive keys are used as the keys of the gate label encryption whereas in the latter case, the inactive keys are being encrypted). Hence, we do not need these inactive keys to construct the rest of the distribution H^n_t (or H^n_{t+1}), as i and j are output wires of gates that precedes t and therefore are "fake" gates. This is the reason for which we had to sort the gates. On the other hand, the process P_n must output the inactive keys of the output wires of the gate $y_1, \ldots, y_m$, since these wires might enter as inputs to another gate $t' > t$ which is a "real" gate in both H^n_t and H^n_{t+1}.

Deriving a contradiction

We now define a related experiment P'_n in which some of the randomness used by P_n is fixed. Specifically, we fix the random strings $W_{y_1}^{b_{y_1}}$, $W_{y_1}^{1-b_{y_1}}, \ldots, W_{y_m}^{b_{y_m}}, W_{y_m}^{1-b_{y_m}}$, c_i, c_j, $c_{y_1}, \ldots, c_{y_m}$ to some value such that the resulting experiments still can be distinguished by A'_n with advantage $\frac{\Delta(n)}{\Gamma(n)}$. (The existence of such strings is promised by an averaging argument.) For simplicity, we omit the fixed strings from the output of this new experiment. The experiments $P'_n(0)$ and $P'_n(1)$ can still be distinguished by some polynomial size circuit with advantage $\frac{\Delta(n)}{\Gamma(n)}$. (Such distinguisher can be constructed by incorporating the omitted fixed strings into A'_n.) Hence, by the contradiction hypothesis, it follows that this advantage is greater than $\frac{1}{\Gamma(n)p(n)}$ for infinitely many n's. As $\Gamma(n)$ is polynomial in n (since C_n is of polynomial size) we deduce that the distribution ensembles $\{P'_n(0)\}_{n\in\mathbb{N}}$ and $\{P'_n(1)\}_{n\in\mathbb{N}}$ are not computationally indistinguishable, in contradiction with the following claim.

Claim 5.2.21 $\{P'_n(0)\}_{n\in\mathbb{N}} \stackrel{c}{\equiv} \{P'_n(1)\}_{n\in\mathbb{N}}$.

Proof: Fix some n. For both distributions $P'_n(0)$ and $P'_n(1)$, the first two entries (i.e., $W_i^{b_i}, W_j^{b_j}$) are two uniformly and independently $2k(n)$-length strings, and the active label $Q_t^{b_i,b_j}$ is a function of $W_i^{b_i}, W_j^{b_j}$ and the fixed strings. Hence, the distributions $P'_n(0)$ and $P'_n(1)$ differ only in the inactive labels $Q_t^{a_i,a_j}$ for $(a_i, a_j) \neq (c_i, c_j)$. In $P'_n(0)$ each of these entries is an all-zeros string that was encrypted under uniformly and independently chosen key R_{a_i,a_j}. In the second distribution $P'_n(1)$, the entry $Q_t^{a_i,a_j}$ is an encryption of a "meaningful" message that was encrypted under the key $W_i^{a_i-r_i,a_j} \oplus W_j^{a_j-r_j,a_i}$, since $(a_i,a_j) \neq (c_i,c_j)$ at least one of the strings $W_i^{a_i-r_i,a_j}, W_j^{a_j-r_j,a_i}$ is not given in the output of $P'_n(1)$ as part of $W_i^{b_i}, W_j^{b_j}$. Also, each of the strings $W_i^{a_i-r_i,a_j}, W_j^{a_j-r_j,a_i}$ was chosen uniformly and it appears only in $Q_t^{a_i,a_j}$ and not in any of the other gate labels, therefore the key $W_i^{a_i-r_i,a_j} \oplus W_j^{a_j-r_j,a_i}$ is distributed uniformly and independently of the other entries of $P'_n(1)$'s output. So all the entries of both $P'_n(1)$ and $P'_n(0)$ are independent. Moreover, the security of the encryption scheme implies that the ensemble $\{Q_t^{a_i,a_j}\}$ for $(a_i,a_j) \neq (c_i,c_j)$ produced by $P'_n(1)$ is computationally indistinguishable from the corresponding ensemble produced by $P'_n(0)$, as in both cases some $p(n)$-length message is encrypted under uniformly chosen $k(n)$-length key. (Recall that $k(n)$ is polynomial in n by definition, and $p(n) = O(|C_n|k(n)) = \mathrm{poly}(n)$). Hence, by Fact 2.2.7, the proof follows. ■

5.3 Applications

5.3.1 Relaxed Assumptions for Cryptography in NC^0

In Chapter 4 we showed that computational randomized encoding preserves the security of many cryptographic primitives.[11] It follows from Theorem 5.2.15 that, under the EPRG assumption, any such primitive can be computed in NC^0 *if it exists at all* (i.e., can be computed in polynomial time). Formally, we have:

Theorem 5.3.1 *Suppose that the EPRG assumption holds. Then,*

1. *If there exists a public-key encryption scheme (resp., NIZK with an efficient prover or constant-round ZK proof with an efficient prover for every* NP *relation), then there exists such a scheme in which the encryption (prover) algorithm is in* NC^0_4.
2. *If there exists a non-interactive commitment scheme, then there exists such a scheme in which the sender is in* NC^0_4 *and the receiver is in* $\mathrm{AND}_n \circ \mathrm{NC}^0$.
3. *There exists a stateless symmetric encryption scheme (resp., digital signature, MAC, a constant-round ZK argument for every language in* NP*) in which the encryption (signing, prover) algorithm is in* NC^0_4.
4. *There exists a constant-round commitment scheme in which the sender is in* NC^0_4 *and the receiver is in* $\mathrm{AND}_n \circ \mathrm{NC}^0$.
5. *For every polynomial-time computable function we have a (non-adaptive single-oracle) IHS in which the user is in* NC^0_5 *and the oracle is in* BPP.

Note that the existence of (stateless) symmetric encryption, signature, MAC, constant-round commitment scheme and constant-round ZK arguments for NP, does not require any additional assumption other than EPRG. This is a consequence of the fact that they all can be constructed (in polynomial time) from a PRG (see [Gol01a, Gol04]). For these primitives, we obtain more general (unconditional) results in the next subsection.

[11]In some cases, we will need to rely on *perfect* correctness, which we get "for free" in our main construction. See Table 4.9.1.

Remark 5.3.2 Theorem 5.3.1 reveals an interesting phenomenon. It appears that several cryptographic primitives (e.g., symmetric encryption schemes, digital signatures and MACs) can be implemented in NC^0 despite the fact that their standard constructions rely on pseudorandom functions (PRFs) [GGM86], which cannot be computed even in AC^0 [LMN93]. For such primitives, we actually construct a sequential PRF from the PRG (as in [GGM86]), use it as a building block to obtain a sequential construction of the desired primitive (e.g., symmetric encryption), and finally reduce the parallel-time complexity of the resulting function using our machinery. Of course, the security of the PRF primitive itself is not inherited by its computational (or even perfect) encoding.

Remark 5.3.3 (Concrete assumptions) Theorem 5.3.1 allows an NC^0 implementations of many primitives under several (new) concrete assumptions. In particular, since the EPRG assumption is implied by the intractability of factoring, the discrete logarithm problem, and lattice problems (see Remark 4.5.8), we can use any of these assumptions to obtain a digital signature (and MAC) whose signing algorithm is in NC^0_4, as well as a non-interactive commitment scheme in which the sender is in NC^0_4 and the receiver is in $\mathrm{AND}_n \circ \mathrm{NC}^0$. Recall that in Section 4.8.2 we showed that, under the same assumptions, there exists a two-round statistically hiding commitment in which the sender is in NC^0_4 and the receiver is in $\mathrm{AND}_n \circ \mathrm{NC}^0$. Hence, by combining the two aforementioned commitments with the ZK protocol of [GK96] (as explained in Section 4.8.3) we get a constant-round ZK-proof for NP between an AC^0 prover and an AC^0 verifier assuming any of the above assumptions. We can also use Theorem 5.3.1 to obtain a NIZK with an NC^0_4 prover for NP under the intractability of factoring. (This follows from the NIZK construction of [FLS00] which can be based on the intractability of factoring [BY96, Gol04, Section C.4.1].)

Parallelizing the receiver

As mentioned above, the computational encoding promised by Theorem 5.2.15 does not support parallel decoding. Thus, we get primitives in which the sender (i.e., the encrypting party, committing party, signer or prover, according to the case) is in NC^0 but the receiver (the decrypting party or verifier) is not known to be in NC, even if we started with a primitive that has an NC receiver. The following theorem tries to partially remedy this state of affairs. Assuming the existence of a PRG with a good stretch in NC^1, we can rely on Claim 5.2.18 to convert sender-receiver schemes in which both the receiver and the sender are in NC to ones in which the sender is in NC^0 and the receiver is still in NC.[12]

Theorem 5.3.4 *Let $\mathcal{X} = (G, S, R)$ be a sender-receiver cryptographic scheme whose security is respected by computational encoding (e.g., encryption, signature, MAC, commitment scheme, NIZK), where G is a key-generation algorithm (in case the scheme has one), $S \in \mathrm{NC}^s$ is the algorithm of the sender and $R \in \mathrm{NC}^r$ is the algorithm of the receiver. Then,*

- *If there exists a polynomial-stretch PRG in NC^1, then there exists a similar scheme $\hat{\mathcal{X}} = (G, \hat{S}, \hat{R})$ in which $\hat{S} \in \mathrm{NC}^0$ and $\hat{R} \in \mathrm{NC}^{\max\{s+1,r\}}$.*
- *If there exists a linear-stretch PRG in NC^1, then there exists a a similar scheme $\hat{\mathcal{X}} = (G, \hat{S}, \hat{R})$, in which $\hat{S} \in \mathrm{NC}^0$ and $\hat{R} \in \mathrm{NC}^{\max\{s+2,r\}}$.*

[12]Similarly, assuming a linear-stretch PRG in NC^1, we can obtain, for every NC function, a (non-adaptive single-oracle) IHS in which the user is in NC^0 and the oracle is in NC.

Proof: If there exists a polynomial-stretch (resp. linear-stretch) PRG in NC^1, then we can use Claim 5.2.18 and get a computational encoding $\hat{S}$ for S in NC^0 whose decoder B is in NC^{s+1} (resp. NC^{s+2}). As usual, the new receiver $\hat{R}$ uses B to decode the encoding, and then applies the original receiver R to the result. Thus, $\hat{R}$ is in $\mathrm{NC}^{\max\{s+1,r\}}$ (resp. $\mathrm{NC}^{\max\{s+2,r\}}$). ■

5.3.2 Parallel Reductions between Cryptographic Primitives

In the previous section we showed that many cryptographic tasks can be performed in NC^0 if they can be performed at all, relying on the assumption that an easy PRG exists. Although EPRG is a very reasonable assumption, it is natural to ask what types of parallel reductions between primitives can be guaranteed *unconditionally.* In particular, such reductions would have consequences even if there exists a PRG in, say, NC^4.

In this section, we consider the types of unconditional reductions that can be obtained using the machinery of Section 5.2. We focus on primitives that can be reduced to a PRG (equivalently, using [HILL99], to a one-way function). We argue that for any such primitive $\mathcal{F}$, its polynomial-time reduction to a PRG can be collapsed into an NC^0-reduction to a PRG. More specifically, we present an efficient "compiler" that takes the code of an arbitrary PRG G and outputs a description of an NC^0 circuit C, having oracle access to a function G', such that for any (minimal) PRG G' the circuit $C[G']$ implements $\mathcal{F}$.

A compiler as above proceeds as follows. Given the code of G, it first constructs a code for an efficient implementation f of $\mathcal{F}$. (In case we are given an efficient *black-box* reduction from $\mathcal{F}$ to a PRG, this code is obtained by plugging the code of G into this reduction.) Then, applying a constructive form of Theorem 5.2.13 to the code of f, the compiler obtains a code $\hat{f}$ of an NC^0 circuit which implements $\mathcal{F}$ by making an oracle access to a PRG. This code of $\hat{f}$ defines the required NC^0 reduction from $\mathcal{F}$ to a PRG, whose specification depends on the code of the given PRG G. Thus, the reduction makes a *non-black-box* use of the PRG primitive, even if the polynomial-time reduction it is based on is fully black-box.

Based on the above we can obtain the following informal "meta-theorem":

Meta-Theorem 5.3.5 *Let $\mathcal{F}$ be a cryptographic primitive whose security is respected by computational encoding. Suppose that $\mathcal{F}$ is polynomial-time reducible to a PRG. Then, $\mathcal{F}$ is NC^0-reducible to a (minimal) PRG.*

Since a minimal PRG can be reduced in NC^0 to one-way permutations or more general types of one-way functions (see Remark 4.5.9), the minimal PRG in the conclusion of the above theorem can be replaced by these primitives.

Instantiating $\mathcal{F}$ by concrete primitives, we get the following corollary:

Corollary 5.3.6 *Let G be a PRG. Then,*

- *There exists a stateless symmetric encryption scheme (resp., digital signature or MAC) in which the encryption (signing) algorithm is in $\mathrm{NC}^0[G]$.*
- *There exists a constant-round commitment scheme (resp., constant-round coin-flipping protocol) in which the sender (first party) is in $\mathrm{NC}^0[G]$ and the receiver (second party) is in $\mathrm{AND}_n \circ \mathrm{NC}^0[G]$.*

- *For every* NP *language, there exists a constant-round ZK argument in which the prover is in* $\mathrm{NC}^0[G]$.

An illustration of the above appears in Figure 1.2.1. Note that items 3,4 of Theorem 5.3.1 can be derived from the above corollary, up to the exact locality.

Comparison with known reductions

The above results can be used to improve the parallel complexity of some known reductions. For example, Naor [Nao91] shows a commitment scheme in which the sender is in $\mathrm{NC}^0[LG]$, where LG is a *linear-stretch* PRG. By using his construction, we derive a commitment scheme in which the sender (respectively, the receiver) is in $\mathrm{NC}^0[G]$ (respectively, $\mathrm{AND}_n \circ \mathrm{NC}^0[G]$) where G is a *minimal* PRG. Since it is not known how to reduce a linear-stretch PRG to a minimal PRG even in NC, we get a nontrivial parallel reduction.

Other interesting examples arise in the case of primitives that are based on PRFs, such as MACs, symmetric encryption, and identification (see [GGM86, NR99, Gol04] for these and other applications of PRFs). Since the known construction of a PRF from a PRG is sequential [GGM86], it was not known how to reduce these primitives in parallel to (even a polynomial-stretch) PRG.[13] This fact motivated the study of parallel constructions of PRFs in [NR99, NR04]. In particular, Naor and Reingold [NR99] introduce a new cryptographic primitive called a synthesizer (SYNTH), and show that PRFs can be implemented in $\mathrm{NC}^1[\mathrm{SYNTH}]$. This gives an NC^1-reduction from cryptographic primitives such as symmetric encryption to synthesizers. By Corollary 5.3.6, we get that these primitives are in fact NC^0-reducible to a PRG. Since (even a polynomial-stretch) PRG can be implemented in $\mathrm{NC}^0[\mathrm{SYNTH}]$ while synthesizers are not known to be even in NC[PRG], our results improve both the complexity of the reduction and the underlying assumption. It should be noted, however, that our reduction only improves the parallel-time complexity of the encrypting party, while the constructions of [NR99] yield NC^1-reductions on both ends.

In contrast to the above, we show that a synthesizer in NC^i can be used to implement encryption in NC^i with decryption in NC.[14] First, we use [NR99] to construct an encryption scheme (E, D) and a polynomial-stretch PRG G such that E and D are in $\mathrm{NC}^1[\mathrm{SYNTH}] = \mathrm{NC}^{i+1}$ and G is in $\mathrm{NC}^0[\mathrm{SYNTH}] = \mathrm{NC}^i$. Next, by Claim 5.2.7 and Remark 5.2.10, we obtain an $\mathrm{NC}^0[G] = \mathrm{NC}^i$ computational encoding $\hat{E}$ for E whose decoder B is in NC^{2i}. (We first use Claim 5.2.7 to construct one-time symmetric encryption (OE, OD) such that OE and OD are in $\mathrm{NC}^0[G] = \mathrm{NC}^i$. Then, we encode E by plugging OE into Construction 5.2.8 and obtain an $\mathrm{NC}^0[OE] = \mathrm{NC}^i$ computational encoding $\hat{E}$ for E. By Remark 5.2.10 the decoder B of $\hat{E}$ is in $\mathrm{NC}^i[OD] = \mathrm{NC}^{2i}$.) To decrypt ciphers of $\hat{E}$ we invoke the decoder B, and then apply the original decryption algorithm D to the result. Therefore, the decryption algorithm of our new scheme $\hat{D}$ is in $\mathrm{NC}^{\max(2i,i+1)}$.

Comparison with the reductions of Chapter 4

Some NC^0 reductions between cryptographic primitives were also obtained in Chapter 4 (see Remarks 4.5.9,4.8.1). In particular, there we considered the following scenario: Let $\mathcal{G}$ be a primitive

[13] Assuming that factoring is intractable (or, more generally, that there exists a PRG in $\oplus\mathrm{L}/poly$) it is provably impossible to reduce PRFs to (sublinear stretch) PRGs in NC^0. See Section 4.9.

[14] For concreteness, we refer here only to the case of symmetric encryption, the case of other primitives which are NC^0-reducible to a PRF (such as identification schemes and MACs) is analogous.

whose security is preserved by randomized encoding. Suppose that $G(x) = g(x, f(q_1(x)), \ldots, f(q_m(x)))$ defines a black-box construction of a primitive G of type $\mathcal{G}$ from a primitive f of type $\mathcal{F}$ where g is in $\mathcal{SREN}$ (or $\mathcal{PREN}$) and the q_i's are in NC^0. (The functions $g, q_1, ..., q_m$ are fixed by the reduction and do not depend on f.) Then, letting $\hat{g}((x, y_1, \ldots, y_m), r)$ be an NC^0 encoding of g, the function $\hat{G}(x, r) = \hat{g}((x, f(q_1(x)), \ldots, f(q_m(x))), r)$ encodes G, and hence defines a (non-adaptive) black-box NC^0 reduction from $\mathcal{G}$ to $\mathcal{F}$. An important example for such a reduction is the transformation of OWF to PRG from [HILL99, Const. 7.1] applied to a *regular* OWF (see Remark 4.5.9). In this case, the transformation is in NC^1 and thus is improved to be in NC^0.[15] Unlike these previous reductions, the current results are not restricted to non-adaptive black-box reductions which are computable in $\mathcal{SREN}$. However, the reductions of this chapter are inherently non black-box and their structure depends on the code implementing the given oracle.

5.3.3 Secure Multi-Party Computation

Secure multi-party computation (MPC) allows several parties to evaluate a function of their inputs in a distributed way, so that both the privacy of their inputs and the correctness of the outputs are maintained. These properties should hold, to the extent possible, even in the presence of an adversary who may corrupt at most t parties. This is typically formalized by comparing the adversary's interaction with the *real process*, in which the uncorrupted parties run the specified protocol on their inputs, with an ideal function evaluation process in which a trusted party is employed. The protocol is said to be *secure* if whatever the adversary "achieves" in the real process it could have also achieved by corrupting the ideal process. A bit more precisely, it is required that for every adversary A interacting with the real process there is an adversary A' interacting with the ideal process, such that outputs of these two interactions are *indistinguishable* from the point of view of an external environment. See, e.g., [Can00, Can01, Gol04], for more detailed and concrete definitions.

There is a variety of different models for secure computation. These models differ in the power of the adversary, the network structure, and the type of "environment" that tries to distinguish between the real process and the ideal process. In the *information-theoretic* setting, both the adversary and the distinguishing environment may be computationally unbounded, whereas in the *computational* setting they are both bounded to probabilistic polynomial time.

The notion of randomizing polynomials was originally motivated by the goal of minimizing the round complexity of MPC. The motivating observation of [IK00] was that the round complexity of most general protocols from the literature (e.g., those of [GMW87, BOGW88, CCD88]) is related to the algebraic *degree* of the function being computed. Thus, by reducing the task of securely computing f to that of securely computing some related low-degree function, one can obtain round-efficient protocols for f.

Randomizing polynomials (or low-degree randomized encodings) provide precisely this type of reduction. More specifically, suppose that the input x to f is distributed between the parties, who wish to all learn the output $f(x)$. If f is represented by a vector $\hat{f}(x, r)$ of degree-d randomizing polynomials, then the secure computation of f can be *non-interactively* reduced to that of $\hat{f}$, where the latter is viewed as a *randomized* function of x. This reduction only requires each party to invoke the decoder of $\hat{f}$ on its local output, obtaining the corresponding output of f. The secure

[15]Similar examples are the NC^1 transformation of one-to-one OWF to non-interactive commitment scheme (cf. [Blu83]) and of distributionally OWF into standard OWF (cf. [IL89]).

computation of $\hat{f}$, in turn, can be non-interactively reduced to that of a related *deterministic* function $\hat{f}'$ of the same degree d. The idea is to let $\hat{f}'(x, r^1, \ldots, r^{t+1}) \overset{\text{def}}{=} p(x, r^1 \oplus \ldots \oplus r^{t+1})$ (where t is a bound on the number of corrupted parties), assign each input vector r^j to a distinct player, and instruct it to pick it at random. (See [IK00] for more details.) This second reduction step is also non-interactive. Thus, any secure protocol for $\hat{f}'$ or $\hat{f}$ gives rise to a secure protocol for f with the same number of rounds. The non-interactive nature of the reduction makes it insensitive to almost all aspects of the security model.

Previous constructions of (perfect or statistical) randomizing polynomials [IK00, IK02, CFIK03] provided *information-theoretic* reductions of the type discussed above. In particular, if the protocol used for evaluating $\hat{f}'$ is information-theoretically secure, then so is the resulting protocol for f. The main limitation of these previous reductions is that they efficiently apply only to restricted classes of functions, typically related to different log-space classes. This situation is remedied in the current work, where we obtain (under the EPRG assumption) a *general* secure reduction from a function f to a related degree-3 function $\hat{f}'$. The main price we pay is that the security of the reduction is no longer information-theoretic. Thus, even if the underlying protocol for $\hat{f}'$ is secure in the information-theoretic sense, the resulting protocol for f will only be computationally secure.

To formulate the above we need the following definitions.

Definition 5.3.7 (Secure computation) *Let $f(x_1, \ldots, x_n)$ be an m-party functionality, i.e., a (possibly randomized) mapping from m inputs of equal length into m outputs. Let π be an m-party protocol. We formulate the requirement that π securely computes f by comparing the following "real process" and "ideal process".*

The real process. *A* t-bounded *adversary A attacking the* real process *is a probabilistic polynomial-time algorithm, who may corrupt up to t parties and observe all of their internal data. At the end of the interaction, the adversary may output an arbitrary function of its view, which consists of the inputs, the random coin tosses, and the incoming messages of the corrupted parties. We distinguish between* passive *vs.* active *adversaries and between* adaptive *vs.* non-adaptive *adversaries. If the adversary is active, it has full control over the messages sent by the corrupted parties, whereas if it is passive, it follows the protocol's instructions (but may try to deduce information by performing computations on observed data). When the set of corrupted parties has to be chosen in advance, we say that the adversary is non-adaptive, and otherwise say that it is adaptive. Given an m-tuple of inputs $(x_1, \ldots, x_m) \in (\{0,1\}^n)^m$, the* output of the real process *is defined as the random variable containing the* concatenation *of the adversary's output with the outputs and identities of the uncorrupted parties. We denote this output by $\textsc{Real}_{\pi,A}(x_1, \ldots, x_m)$.*

The ideal process. *In the ideal process, an incorruptible trusted party is employed for computing the given functionality. That is, the "protocol" in the ideal process instructs each party to send its input to the trusted party, who computes the functionality f and sends to each party its output. The interaction of a t-bounded adversary A' with the ideal process and the output of the ideal process are defined analogously to the above definitions for the real process. The adversary attacking the ideal process will also be referred to as a* simulator*. We denote the output of the ideal process on the inputs $(x_1, \ldots, x_m) \in (\{0,1\}^n)^m$ by $\textsc{Ideal}_{f,A'}(x_1, \ldots, x_m)$.*

The protocol π is said to t-securely *realize the given functionality f with respect to a specified type of adversary (namely, passive or active, adaptive or non-adaptive) if for any probabilis-*

tic polynomial-time t-bounded adversary A attacking the real process, there exists a probabilistic polynomial-time t-bounded simulator A' attacking the ideal process, such that for any sequence of m-tuples $\{\bar{x}_n\}$ such that $\bar{x}_n \in (\{0,1\}^n)^m$, it holds that $\textsc{Real}_{\pi,A}(\bar{x}_n) \overset{c}{\equiv} \textsc{Ideal}_{f,S}(\bar{x}_n)$.

Secure reductions. To define secure reductions, consider the following *hybrid* model. An m-party protocol augmented with an oracle to the m-party functionality g is a standard protocol in which the parties are allowed to invoke g, i.e., a trusted party to which they can securely send inputs and receive the corresponding outputs. The notion of t-security generalizes to protocols augmented with an oracle in the natural way.

Definition 5.3.8 *Let f and g be m-party functionalities. A* t-secure *reduction from f to g is an m-party protocol that given an oracle access to the functionality g, t-securely realizes the functionality f (with respect to a specified type of adversary). We say that the reduction is* non-interactive *if it involves a single call to f (and possibly local computations on inputs and outputs), but no further communication.*

Appropriate composition theorems, (e.g. [Gol04, Thms. 7.3.3, 7.4.3]), guarantee that the call to g can be replaced by any secure protocol realizing g, without violating the security of the high-level protocol for f.[16] Using the above terminology, Theorem 5.2.15 has the following corollary.

Theorem 5.3.9 *Suppose the EPRG assumption holds. Let $f(x_1, \ldots, x_m)$ be an m-party functionality computed by a (uniform) circuit family of size $s(n)$. Then, for any $\epsilon > 0$, there is a non-interactive, computationally $(m-1)$-secure reduction from f to either of the following two efficient functionalities:*

- *A randomized functionality $\hat{f}(x_1, \ldots, x_m)$ of degree 3 (over $\mathbb{F}_2$) with a random input and output of length $O(s(n) \cdot n^\epsilon)$ each;*
- *A deterministic functionality $\hat{f}'(x'_1, \ldots, x'_m)$ of degree 3 (over $\mathbb{F}_2$) with input length $O(m \cdot s(n) \cdot n^\epsilon)$ and output length $O(s(n) \cdot n^\epsilon)$.*

Both reductions are non-interactive in the sense that they involve a single call to $\hat{f}$ or $\hat{f}'$ and no further interaction. They both apply regardless of whether the adversary is passive or active, adaptive or non-adaptive.

Proof: The second item follows from the first via standard (non-interactive, degree-preserving) secure reduction from randomized functionalities to deterministic functionalities (see [Gol04, Prop. 7.3.4]). Thus we will only prove the first item. Assume, without loss of generality, that f is a deterministic functionality that returns the same output to all the parties.[17] Let $\hat{f}(x, r)$ be the computational encoding of $f(x)$ promised by Theorem 5.2.15. (Recall that $\hat{f}$ is indeed a degree

[16]Actually, for the composition theorem to go through, Definition 5.3.7 should be augmented by providing players and adversaries with auxiliary inputs. We ignore this technicality here, and note that the results in this section apply (with essentially the same proofs) to the augmented model as well.

[17]To handle randomized functionalities we use the non-interactive secure reduction mentioned above. Now, we can $(m-1)$-securely reduce f to a single-output functionality by letting each party to mask its output f_i with a private randomness. That is, $f'((x_1, r_1) \ldots, (x_m, r_m)) = ((f_1(x_1) \oplus r_1) \circ \ldots \circ (f_1(x_m) \oplus r_m))$. As both reductions are non-interactive the resulting reduction is also non-interactive. Moreover, the circuit size of f' is linear in the size of the circuit that computes the original function.

3 function having $O(s(n)\cdot n^{\epsilon})$ random inputs and outputs.) The following protocol $(m-1)$-securely reduces the computation of $f(x)$ to $\hat{f}(x,r)$ (where $\hat{f}$ is viewed as a randomized functionality whose randomness is r).

- Inputs: Party i gets input $x_i \in \{0,1\}^n$.
- Party i invokes the (randomized) oracle $\hat{f}$ with query x_i, and receives an output $\hat{y}$.
- Outputs: Each party locally applies the decoder B of the encoding to the answer $\hat{y}$ received from the oracle, and outputs the result.

We start by showing that the reduction is $(m-1)$-secure against a passive non-adaptive adversary. Let A be such an adversary that attacks some set $I \subset [m]$ of the players. Then, the output of the real process is $(A(x_I, \hat{f}(x,r)), B(\hat{f}(x,r)), \bar{I})$ where $x_I = (x_i)_{i\in I}$, $\bar{I} \stackrel{\text{def}}{=} [m] \setminus I$ and r is a uniformly chosen string of an appropriate length. We define a (passive non-adaptive) simulator A' that attacks the ideal process in the natural way: that is, $A'(x_I, y) = A(x_I, S(y))$, where y is the answer received from the trusted party (i.e., $f(x)$) and S is the computationally private simulator of the encoding. Thus, the output of the ideal process is $(A'(x_I, f(x)), f(x), \bar{I})$. By the definition of A', the privacy of the encoding $\hat{f}$ and Fact 2.2.8, we have,

$$\textsc{Ideal}(x) \equiv (A(x_I, S(f(x))), f(x), \bar{I}) \stackrel{c}{\equiv} (A(x_I, \hat{f}(x,r)), B(\hat{f}(x,r)), \bar{I}) \equiv \textsc{Real}(x),$$

which finishes the proof.

We now sketch the security proof for the case of an adversary A which is both adaptive and active. (The non-adaptive active case as well as the adaptive passive case are treated similarly.) An attack by A has the following form: (1) Before calling the oracle $\hat{f}$, in each step A may decide (according to his current view) to corrupt some party i and learn its input x_i. (2) When the oracle $\hat{f}$ is invoked A changes the input of each corrupted party i to some value x'_i, which is handed to the $\hat{f}$ oracle. (3) After the parties call the oracle on some (partially corrupted) input $x' = (x'_I, x_{\bar{I}})$, the oracle returns a randomized encoding $\hat{f}(x')$ to the adversary, and now A may adaptively corrupt additional parties. Finally, A outputs some function of its entire view. For every such adversary we construct a simulator A' that attacks the ideal process by invoking A and emulating his interaction with the real process. Namely, (1) Before the call to the trusted party we let A choose (in each step) which party to corrupt and feed it with the input we learn; (2) When the trusted party is invoked we let A pick x'_I according to its current view and send these x'_I to the f oracle; (3) Given the result $y = f(x'_I, x_{\bar{I}})$ returned by the oracle, we invoke the simulator (of the encoding) on y and feed the result to A. Finally, we let A pick new corrupted players as in step (1). We claim that in each step of the protocol the view of A when interacting with the real process is computationally indistinguishable from the view of A when it is invoked by A' in the ideal process. (In fact, before the call to the oracle these views are identically distributed.) Hence, the outputs of the two processes are also computationally indistinguishable. ■

A high-level corollary of Theorem 5.3.9 is that computing arbitrary polynomial-time computable functionalities is as easy as computing degree-3 functionalities. Thus, when designing new MPC protocols, it suffices to consider degree-3 functionalities which are often easier to handle.

More concretely, Theorem 5.3.9 gives rise to new, conceptually simpler, constant-round protocols for general functionalities. For instance, a combination of this result with the "BGW protocol" [BOGW88] gives a simpler alternative to the constant-round protocol of Beaver, Micali, and

Rogaway [BMR90]. The resulting protocol will be more round-efficient, and in some cases (depending on the number of parties and the "easiness" of the PRG) even more communication-efficient than the protocol of [BMR90]. On the downside, Theorem 5.3.9 relies on a stronger assumption than the protocol from [BMR90] (an easy PRG vs. an arbitrary PRG).

An interesting open question, which is motivated mainly from the point of view of the MPC application, is to come up with an "arithmetic" variant of the construction. That is, given an arithmetic circuit C, say with addition and multiplication gates, construct a vector of computationally private randomizing polynomials of size poly($|C|$) which makes a *black-box* use of the underlying field. The latter requirement means that the same polynomials should represent C over any field, ruling out the option of simulating arithmetic field operations by boolean operations. Such a result is known for weaker arithmetic models such as formulas and branching programs (see [CFIK03]).

Chapter 6

On Pseudorandom Generators with Linear Stretch in NC^0

Summary: We consider the question of constructing cryptographic pseudorandom generators in NC^0 with large stretch. Our previous constructions of such PRGs were limited to stretching a seed of n bits to $n + o(n)$ bits. This leaves open the existence of a PRG with a linear (let alone superlinear) stretch in NC^0. In this chapter we study this question and obtain the following main results:

1. We show that the existence of a linear-stretch PRG in NC^0 implies non-trivial hardness of approximation results *without relying on PCP machinery.* In particular, it implies that Max3SAT is hard to approximate to within some multiplicative constant.

2. We construct a linear-stretch PRG in NC^0 under a specific intractability assumption related to the hardness of decoding "sparsely generated" linear codes. Such an assumption was previously conjectured by Alekhnovich [Ale03].

6.1 Introduction

A cryptographic pseudorandom generator (PRG) (cf. Definition 4.5.1) is a deterministic function that stretches a short random seed into a longer string that cannot be distinguished from random by any polynomial-time observer. In this chapter, we study the existence of PRGs that both (1) admit fast parallel computation and (2) stretch their seed by a significant amount.

Considering the first goal alone, we showed in Chapter 4 that the ultimate level of parallelism can be achieved under most standard cryptographic assumptions. Specifically, any PRG in $\oplus\mathrm{L}/poly$ (the existence of which follows, for example, from the intractability of factoring) can be efficiently "compiled" into a PRG in NC^0. However, the PRGs produced by this compiler only stretch their seed by a sublinear amount: from n bits to $n + O(n^{\varepsilon})$ bits for some constant $\varepsilon < 1$. Thus, these PRGs do not meet our second goal.

Considering the second goal alone, even a PRG that stretches its seed by just one bit can be used to construct a PRG that stretches its seed by any polynomial number of bits [Gol01a, Sec. 3.3.2]. However, all known constructions of this type are inherently sequential. Thus, we cannot use known techniques for turning an NC^0 PRG with a sublinear stretch into one with a linear, let alone superlinear, stretch.

The above state of affairs leaves open the existence of a *linear-stretch* PRG (LPRG) in NC^0; namely, one that stretches a seed of n bits into $n+\Omega(n)$ output bits.[1] (In fact, there was no previous evidence for the existence of LPRGs even in the higher complexity class AC^0.) This question is the main focus of this chapter. The question has a very natural motivation from a cryptographic point of view. Indeed, most cryptographic applications of PRGs either require a linear stretch (for example Naor's bit commitment scheme [Nao91][2]), or alternatively depend on a larger stretch for efficiency (this is the case for the standard construction of a stream cipher or stateful symmetric encryption from a PRG, see [Gol04]). Thus, the existence of an LPRG in NC^0 would imply better parallel implementations of other cryptographic primitives.

6.1.1 Our Contribution

LPRG in NC^0 implies hardness of approximation. We give a very different, and somewhat unexpected, motivation for the foregoing question. We observe that the existence of an LPRG in NC^0 *directly* implies non-trivial and useful hardness of approximation results. Specifically, we show (via a simple argument) that an LPRG in NC^0 implies that Max3SAT (and hence all MaxSNP problems such as Max-Cut, Max2SAT and Vertex Cover [PY91]) cannot be efficiently approximated to within some multiplicative constant. This continues a recent line of work, initiated by Feige [Fei02] and followed by Alekhnovich [Ale03], that provides simpler alternatives to the traditional PCP-based approach by relying on stronger assumptions. Unlike these previous works, which rely on very specific assumptions, our assumption is of a more general flavor and may serve to further motivate the study of cryptography in NC^0. On the down side, the conclusions we get are weaker and in particular are implied by the PCP theorem. In contrast, some inapproximability results from [Fei02, Ale03] could not be obtained using PCP machinery. It is instructive to note that by applying our general argument to the sublinear-stretch PRGs in NC^0 from Chapter 4 we only get "uninteresting" inapproximability results that follow from standard padding arguments (assuming $\mathrm{P} \neq \mathrm{NP}$). Furthermore, we do not know how to obtain stronger inapproximability results based on a superlinear-stretch PRG in NC^0. Thus, our main question of constructing LPRGs in NC^0 captures precisely what is needed for this application.

Constructing an LPRG in NC^0. We present a construction of an LPRG in NC^0 under a specific intractability assumption related to the hardness of decoding "sparsely generated" linear codes. Such an assumption was previously made by Alekhnovich in [Ale03]. The starting point of our construction is a modified version of a PRG from [Ale03] that has a large output locality (that is, each output bit depends on many input bits) but has a simple structure. We note that the output distribution of this generator can be sampled in NC^0; however the seed length of this NC^0 sampling procedure is too large to gain any stretch. To solve this problem we observe that the seed has large entropy even when the output of the generator is given. Hence, we can regain the stretch by employing a randomness extractor in NC^0 that uses a "sufficiently short" seed to extract randomness from sources with a "sufficiently high" entropy. We construct the latter by combining

[1]Recall that an NC^0 LPRG can be composed with itself a constant number of times to yield an NC^0 PRG with an arbitrary linear stretch. See Remark 4.5.3.

[2]In Chapter 5 we showed that there is an NC^0 construction of a commitment scheme from an arbitrary PRG including one with sublinear stretch (see Corollary 5.3.6). However, this construction makes a non-black-box use of the underlying PRG, and is thus quite inefficient. The only known parallel construction that makes a black-box use of the PRG is Naor's original construction, which requires the PRG to have linear stretch.

the known construction of randomness extractors from ε-biased generators [NN93, BSSVW03] with previous constructions of ε-biased generator in NC^0 [MST03]. Our LPRG can be implemented with locality 4; the stretch of this LPRG is essentially optimal, as it is known that no PRG with locality 4 can have a *superlinear* stretch [MST03]. However, the existence of superlinear-stretch PRG with possibly higher (but constant) locality remains open.

By combining the two main results described above, one gets non-trivial inapproximability results under the intractability assumption from [Ale03]. These (and stronger) results were *directly* obtained in [Ale03] from the same assumption *without* constructing an LPRG in NC^0. Our hope is that future work will yield constructions of LPRGs in NC^0 under different, perhaps more standard, assumptions, and that the implications to hardness of approximation will be strengthened.

LPRG in NC^0 and Expanders. Finally, we observe that the input-output graph of any LPRG in NC^0 enjoys some non-trivial expansion property. This connection implies that a (deterministic) construction of an LPRG in NC^0 must use some non-trivial combinatorial objects. (In particular, one cannot hope that "simple" transformations, such as those given in Chapter 4, will yield LPRGs in NC^0.) The connection with expanders also allows to rule out the existence of *exponentially*-strong PRGs with *superlinear* stretch in NC^0.

6.1.2 Related Work

The first application of average-case complexity to inapproximability was suggested by Feige [Fei02], who derived new inapproximability results under the assumption that refuting 3SAT is hard on average on some natural distribution. In [Ale03] Alekhnovich continued this line of research. He considered the problem of determining the maximal number of satisfiable equations in a linear system chosen at random, and made several conjectures regarding the average case hardness of this problem. He showed that these conjectures imply Feige's assumption as well as several new inapproximability results. While the works of Feige and Alekhnovich derived *new* inapproximability results (that were not known to hold under the assumption that $\mathrm{P} \neq \mathrm{NP}$), they did not rely on the relation with a standard cryptographic assumption or primitive, but rather used specific average case hardness assumptions tailored to their inapproximability applications. A relation between the security of a cryptographic primitive and approximation was implicitly used in [MST03], where an approximation algorithm for Max2LIN was used to derive an upper bound on the stretch of a PRG whose locality is 4.

Organization. The rest of this paper is structured as follows. We begin with a discussion of notation and preliminaries (Section 6.2). In Section 6.3 we prove that an LPRG in NC^0 implies that Max3SAT cannot be efficiently approximated to within some multiplicative constant. Then in Section 6.4 we extend these results and show how to derive the inapproximability of Max3SAT from NC^0 implementations of other cryptographic primitives. In Section 6.5 we present a construction of an LPRG in NC^0. This construction uses an NC^0 implementation of an ε-biased generator as an ingredient. A uniform construction of such an ε-biased generator is described in Section 6.5.4. Finally, in Section 6.6, we discuss the connection between LPRG in NC^0 to expander graphs.

6.2 Preliminaries

Pseudorandom generators. Recall that a PRG, $G : \{0,1\}^n \to \{0,1\}^{s(n)}$ is a deterministic function that stretches a short random seed into a longer pseudorandom string (see Definition 4.5.1). When $s(n) = n + \Omega(n)$ we say that G is a *linear-stretch* pseudorandom generator (LPRG). By default, we require G to be polynomial-time computable. It will sometimes be convenient to define a PRG by an infinite family of functions $\{G_n : \{0,1\}^{m(n)} \to \{0,1\}^{s(n)}\}_{n\in\mathbb{N}}$, where $m(\cdot)$ and $s(\cdot)$ are polynomials. Such a family can be transformed into a single function that satisfies Definition 4.5.1 via padding (see Remark 3.1.8). We will also rely on *ε-biased generators*, defined similarly to PRGs except that the pseudorandomness holds only against linear functions over $\mathbb{F}_2$. (See Definition 4.5.10.)

Expanders. In the followings think of m as larger than n. We say that a bipartite graph $G = ((L = [m], R = [n]), E)$ is (K, α) expanding if every set of left vertices S of size smaller than K has at least $\alpha \cdot |S|$ right neighbors. A family of bipartite graphs $\{G_n\}_{n\in\mathbb{N}}$ where $G_n = ((L = [m(n)], R = [n]), E)$ is expanding if for some constants α and β and sufficiently large n the graph G_n is $(\beta \cdot m(n), \alpha)$ expanding. A family of $m(n) \times n$ binary matrices $\{M_n\}_{n\in\mathbb{N}}$ is expanding if the family of bipartite graphs $\{G_n\}_{n\in\mathbb{N}}$ represented by $\{M_n\}_{n\in\mathbb{N}}$ (i.e., M_n is the adjacency matrix of G_n) is expanding.

6.2.1 Some useful facts

We will use the following well known bound on the sum of binomial coefficients.

Fact 6.2.1 *For $0 < p \le 1/2$ we have $\sum_{i=0}^{pn} \binom{n}{i} \le 2^{n\mathrm{H}_2(p)}$.*

The *bias* of a Bernoulli random variable X is defined to be $|\Pr[X = 1] - \frac{1}{2}|$. We will need the following fact which estimates the bias of sum of independent random coins (cf. [MST03, Shp06]).

Fact 6.2.2 *Let $X_1, \ldots, X_t$ be independent binary random variables. Suppose that for some $0 < \delta < \frac{1}{2}$ and every i it holds that $\mathrm{bias}(X_i) \le \delta$. Then, $\mathrm{bias}(\bigoplus_{i=1}^{t} X_i) \le \frac{1}{2}(2\delta)^t$.*

6.3 LPRG in NC^0 implies Hardness of Approximation

In the following we show that if there exists an LPRG in NC^0 then there is no polynomial-time approximation scheme (PTAS) for Max3SAT; that is, Max3SAT cannot be efficiently approximated within some multiplicative constant $r > 1$. Recall that in the Max3SAT problem we are given a 3CNF boolean formula with s clauses over n variables, and the goal is to find an assignment that satisfies the largest possible number of clauses. The Max ℓ-CSP problem is a generalization of Max3SAT in which instead of s clauses we get s boolean constraints $C = \{C_1, \ldots, C_s\}$ of arity ℓ. Again, our goal is to find an assignment that satisfies the largest possible number of constraints. (Recall that a constraint C of arity ℓ over n variables is an ℓ-local boolean function $f : \{0,1\}^n \to \{0,1\}$, and it is satisfied by an assignment $(\sigma_1, \ldots, \sigma_n)$ if $f(\sigma_1, \ldots, \sigma_n) = 1$.)

A simple and useful corollary of the PCP Theorem [ALM+98, AS98] is the inapproximability of Max3SAT.

Theorem 6.3.1 *Assume that $\mathrm{P} \neq \mathrm{NP}$. Then, there is an $\varepsilon > 0$ such that there is no $(1 + \varepsilon)$-approximatation algorithm for Max3SAT.*

We will prove a similar result under the (stronger) assumption that there exists an LPRG in NC^0. Our proof, however, does not rely on the PCP Theorem.

Theorem 6.3.2 *Assume that there exists an LPRG in* NC^0. *Then, there is an* $\varepsilon > 0$ *such that there is no* $(1+\varepsilon)$*-approximation algorithm for Max3SAT.*

The proof of Theorem 6.3.2 follows by combining the following Fact 6.3.3 and Lemma 6.3.4. The first fact shows that in order to prove that Max3SAT is hard to approximate, it suffices to prove that Max ℓ-CSP is hard to approximate. This standard result follows by applying Cook's reduction to transform every constraint into a 3CNF.

Fact 6.3.3 *Assume that, for some constants* $\ell \in \mathbb{N}$ *and* $\varepsilon > 0$, *there is no polynomial time* $(1+\varepsilon)$*-approximation algorithm for Max* ℓ*-CSP. Then there is an* $\varepsilon' > 0$ *such that there is no polynomial time* $(1+\varepsilon')$*-approximation algorithm for Max3SAT.*

Thus, the heart of the proof of Theorem 6.3.2 is showing that the existence of an LPRG in NC^0_ℓ implies that there is no PTAS for Max ℓ-CSP.

Lemma 6.3.4 *Let* ℓ *be a positive integer, and* $c > 1$ *be a constant such that* $G : \{0,1\}^n \rightarrow \{0,1\}^{cn}$ *is an LPRG which is computable in* NC^0_ℓ. *Then, there is no* $1/(1-\varepsilon)$*-approximation algorithm for Max* ℓ*-CSP, where* $0 < \varepsilon < 1/2$ *is a constant that satisfies* $\mathrm{H}_2(\varepsilon) < 1 - 1/c$.

For $\varepsilon = 1/10$ (i.e., ≈ 1.1-approximation) the constant $c = 2$ will do, whereas for $\varepsilon = 0.49$ (i.e., ≈ 2-approximation) $c = 3500$ will do.
Proof: Let $s = s(n) = cn$. Assume towards a contradiction that there exists an $1/(1-\varepsilon)$-approximation algorithm for Max ℓ-CSP where $\mathrm{H}_2(\varepsilon) < 1 - 1/c$. Then, there exists a polynomial-time algorithm A that given an ℓ-CSP instance ϕ outputs 1 if ϕ is satisfiable, and 0 if ϕ is ε-unsatisfiable (i.e., if every assignment fails to satisfy at least a fraction ε of the constraints). We show that, given such A, we can "break" the LPRG G; that is, we can construct an efficient (non-uniform) adversary that distinguishes between $G(U_n)$ and U_s. Our adversary B_n will (deterministically) translate a string $y \in \{0,1\}^s$ into an ℓ-CSP instance ϕ_y with s constraints such that the following holds:

1. If $y \leftarrow G(U_n)$ then ϕ_y is always satisfiable.
2. If $y \leftarrow U_s$ then, with probability $1 - \mathrm{neg}(n)$ over the choice of y, no assignment satisfies more than $(1-\varepsilon)s$ constraints of ϕ_y.

Then, B_n will run A on ϕ_y and will output $A(\phi_y)$. The distinguishing advantage of B is $1 - \mathrm{neg}(n)$ in contradiction to the pseudorandomness of G.

It is left to show how to translate $y \in \{0,1\}^s$ into an ℓ-CSP instance ϕ_y. We use n boolean variables $x_1, \ldots, x_n$ that represent the bits of an hypothetical pre-image of y under G. For every $1 \le i \le s$ we add a constraint $G_i(x) = y_i$ where G_i is the function that computes the i-th output bit of G. Since G_i is an ℓ-local function the arity of the constraint is at most ℓ.

Suppose first that $y \leftarrow G(U_n)$. Then, there exists a string $\sigma \in \{0,1\}^n$ such that $G(\sigma) = y$ and hence ϕ_y is satisfiable. We move on to the case where $y \leftarrow U_s$. Here, we rely on the fact that such a random y is very likely to be far from every element in the range of G. More formally, define a set $\mathrm{BAD}_n \subseteq \{0,1\}^s$ such that $y \in \mathrm{BAD}_n$ if ϕ_y is $(1-\varepsilon)$-satisfiable; that is, if there exists an assignment $\sigma \in \{0,1\}^n$ that satisfies at least $(1-\varepsilon)$ fraction of the constraints of ϕ_y. In other words, the Hamming distance between y and $G(\sigma)$ is at most εs. Hence, all the elements of BAD_n are εs-close (in Hamming distance) to some string in $\mathrm{Im}(G)$. Therefore, the size of BAD_n is bounded by

$$|\mathrm{Im}(G)| \cdot \sum_{i=0}^{\varepsilon s} \binom{s}{i} \leq 2^n 2^{\mathrm{H}_2(\varepsilon)s} = 2^{(1+c\mathrm{H}_2(\varepsilon))n},$$

where the first inequality is due to Fact 6.2.1. Let $\alpha \stackrel{\text{def}}{=} c-(1+c\cdot\mathrm{H}_2(\varepsilon))$ which is a positive constant since $\mathrm{H}_2(\varepsilon) < 1-1/c$. Hence, we have

$$\Pr_{y\leftarrow U_s}[\phi_y \text{ is } (1-\varepsilon) \text{ satisfiable}] = |\mathrm{BAD}_n| \cdot 2^{-s} \leq 2^{(1+c\mathrm{H}_2(\varepsilon))n-cn} = 2^{-\alpha n} = \mathrm{neg}(n),$$

which completes the proof. ■

Remark 6.3.5 Lemma 6.3.4 can tolerate some relaxations to the notion of LPRG. In particular, since the advantage of B_n is exponentially close to 1, we can consider an LPRG that satisfies a weaker notion of pseudorandomness in which the distinguisher's advantage is bounded by $1-1/p(n)$ for some polynomial $p(n)$. In Section 6.4 we consider additional cryptographic primitives that imply the inapproximability of Max3SAT.

Lemma 6.3.4 implies the following corollary.

Corollary 6.3.6 *Suppose there exists a PRG in* NC^0_ℓ *with* an arbitrary *linear stretch; i.e., for every* $c > 0$ *there exists a PRG* $G : \{0,1\}^n \to \{0,1\}^{c\cdot n} \in \mathrm{NC}^0_\ell$. *Then, Max* ℓ*-CSP cannot be approximated to within any constant* $\delta < 2$ *that is arbitrarily close to 2.*

Remark 6.3.7 Corollary 6.3.6 is tight, as any CSP problem of the form $G(x) = y$ (for any $y \in \{0,1\}^s$) can be easily approximated within a factor of 2. To see this, note that the function $G_i(x)$ which computes the i-th output bit of G must be balanced, i.e., $\Pr_x[G_i(x) = 1] = 1/2$. (Otherwise, since $G_i \in \mathrm{NC}^0$, the function G_i has a constant bias and so $G(U_n)$ cannot be pseudorandom.) Therefore, a random assignment is expected to satisfy 1/2 of the constraints of the instance $G(x) = y$. This algorithm can be derandomized by using the method of conditional expectations.

Papadimitriou and Yannakakis [PY91] defined a class MaxSNP, in which Max3SAT is complete in the sense that any problem in MaxSNP has a PTAS if and only if Max3SAT has a PTAS. Hence, we get the following corollary (again, without the PCP machinery):

Corollary 6.3.8 *Assume that there exists an LPRG in* NC^0. *Then, all Max SNP problems (e.g., Max-Cut, Max2SAT, Vertex Cover) do not have a PTAS.*

6.4 Using NC^0 Implementations of Other Cryptographic Primitives

In the following we extend the results of the Section 6.3, and show that the inapproximability of Max3SAT can be based on NC^0 implementations of the following primitives: (1) pseudoentropy generator that gains a linear amount of computational entropy; (2) string commitment of linear size; and (3) public-key encryption whose ciphertext length is linear in the message length. We start by abstracting the proof of Theorem 6.3.2. That is, we show that the following assumption imply the inapproximability of Max3SAT.

Consider a pair of distribution ensembles A and B, a parameter δ, and a constant ε. The assumption holds if (1) A is samplable by NC^0 circuits; (2) the computational distance between A and B is bounded by δ; and (3) the probability that the outcome of B will be ε-close to the support of A is smaller than $1-\delta$. More formally, we assume the following.

Intractability Assumption 6.4.1 *There exist two distribution ensembles $\{A_n\}_{n\in\mathbb{N}}$ and $\{B_n\}_{n\in\mathbb{N}}$ where A_n and B_n are distributed over $\{0,1\}^{s(n)}$, and the ensemble $\{A_n\}$ is samplable by an* NC^0 *circuit family. There exists a function $\delta(n):\mathbb{N}\rightarrow[0,1]$, and a constant $\varepsilon>0$ such that the following holds:*

1. *$\{A_n\} \overset{c}{\equiv}_{\delta(n)} \{B_n\}$. That is, every polynomial-size circuit family distinguishes $\{A_n\}$ from $\{B_n\}$ with advantage at most $\delta(n)$ for sufficiently large n.*

2. *With probability smaller than $1-\delta(n)$ a string $b\leftarrow B_n$ is ε-close (in normalized Hamming distance) to some string in the support of A_n. That is,* $\Pr_{b\leftarrow B_n}[\exists a\in \mathrm{support}(A_n)$ *s.t.* $\mathrm{dist}(a,b)\le \varepsilon\cdot s(n)] < 1-\delta(n)$, *where* $\mathrm{dist}(a,b)$ *denotes the Hamming distance between the strings a and b.*

This assumption is implied by the existence of an LPRG in NC^0. Indeed, if $G:\{0,1\}^n\rightarrow\{0,1\}^{cn}$ is an LPRG in NC^0 then Assumption 6.4.1 holds with respect to $A_n=G(U_n)$, $B_n=U_{cn}$, $\delta(n)=1/n$ and a constant $0<\varepsilon<1/2$ that satisfies $1+c\cdot \mathrm{H}_2(\varepsilon)<c$. (This is implicitly shown in the proof of Lemma 6.3.4.)

Lemma 6.4.2 *Assumption 6.4.1 implies that there is no PTAS for Max3SAT.*

Proof sketch: The proof is very similar to the proof of Lemma 6.3.4. Let $G\in \mathrm{NC}^0_\ell$ be the circuit that samples the distribution A_n. Assume towards a contradiction that the claim does not hold. Then, there exists an algorithm D that given an ℓ-CSP instance ϕ outputs 1 if ϕ is satisfiable, and 0 if ϕ is ε-unsatisfiable. We use this procedure to distinguish $\{A_n\}$ from $\{B_n\}$ with advantage greater than $\delta(n)$. Given a challenge $y\in\{0,1\}^{s(n)}$, we translate it into an ℓ-CSP instance ϕ_y of the form $G(x)=y$, and output $D(\phi_y)$. If $y\leftarrow A_n$ then ϕ_y is always satisfiable. On the other hand, if $y\leftarrow B_n$ then, with probability larger than $\delta(n)$, the formula ϕ_y is ε-unsatisfiable. ■

6.4.1 Pseudoentropy Generator

We now show that an NC^0 implementation of a relaxed notion of LPRG implies Assumption 6.4.1. In particular, instead of being pseudorandom, the distribution $G(U_n)$ is only required to be computationally close to some distribution whose min-entropy is (much) larger. Moreover, we allow a non-negligible distinguishing advantage. This relaxation can be considered as a weak pseudoentropy generator that gains a linear amount of computational entropy cf.[HILL99, BSW03].

Lemma 6.4.3 (Weak LPRG in NC^0 ⇒ inapproximability) *Suppose that there exist an* NC^0 *function $G:\{0,1\}^n\rightarrow\{0,1\}^{s(n)}$ and a distribution ensemble $\{B_n\}$, such that:*

- *$\{G(U_n)\} \overset{c}{\equiv}_{\delta(n)} \{B_n\}$ for some $\delta(n)$ such that $\delta(n) \leq 1 - 2^{-o(s(n))}$.*
- $\mathrm{H}_\infty(B_n) - n = \Omega(s(n))$.

Then, there is no PTAS for Max3SAT.

Proof: Let $A_n \overset{\text{def}}{=} G(U_n)$. We show that $A_n, B_n, \delta(n)$ and some constant $\varepsilon < 1/2$ satisfy Assumption 6.4.1. Indeed, the only non-trivial part is item (2). Let $\mathrm{BAD}_{\varepsilon,n} \subseteq \{0,1\}^{s(n)}$ be the set of all strings which are ε-close to $\mathrm{Im}(G)$. Then,

$$\begin{aligned}
\Pr_{b \leftarrow B_n}[b \text{ is } \varepsilon\text{-close to } \mathrm{Im}(G)] &= \sum_{y \in \mathrm{BAD}_{\varepsilon,n}} \Pr_{b \leftarrow B_n}[b = y] \\
&\leq \left(|\mathrm{Im}(G)| \cdot \sum_{i=0}^{\varepsilon s(n)} \binom{s(n)}{i} \right) \cdot 2^{-\mathrm{H}_\infty(B_n)} \\
&\leq 2^{n + s(n)\cdot \mathrm{H}_2(\varepsilon) - \mathrm{H}_\infty(B_n)} \leq 2^{-\Omega(s(n))} < 1 - \delta(n),
\end{aligned}$$

where the second inequality is due to Fact 6.2.1, the third inequality holds for sufficiently small (constant) ε, and the last inequality holds for sufficiently large n . ■

6.4.2 String Commitment

Another sufficient assumption is an NC^0 implementation of a non-interactive string commitment with a constant information rate, namely one in which the length of the commitment is linear in that of the committed string. A non-interactive commitment scheme is defined by a function $\mathrm{Com} : \{0,1\}^n \times \{0,1\}^{m(n)} \to \{0,1\}^{s(n)}$ such that:

1. (Binding) For every pair of different strings $x, y \in \{0,1\}^n$ the supports of $\mathrm{Com}(x, U_{m(n)})$ and $\mathrm{Com}(y, U_{m(n)})$ are disjoint.
2. (Hiding) For every pair of string families $\{x_n\}_{n \in \mathbb{N}}$ and $\{y_n\}_{n \in \mathbb{N}}$ where $x_n, y_n \in \{0,1\}^n$, we have $\mathrm{Com}(x_n, U_{m(n)}) \overset{c}{\equiv} \mathrm{Com}(y_n, U_{m(n)})$.

In fact, for our purpose we can relax the hiding property to be $\mathrm{Com}(x_n, U_{m(n)}) \overset{c}{\equiv}_{\delta(n)} \mathrm{Com}(y_n, U_{m(n)})$ where $\delta(n) = 1 - 2^{-o(n)}$.

Lemma 6.4.4 (Constant rate string commitment in $\mathrm{NC}^0 \Rightarrow$ inapproximability) *Let $c > 1$ be a constant. Suppose that there exists a (non-interactive) commitment scheme $\mathrm{Com} : \{0,1\}^n \times \{0,1\}^{m(n)} \to \{0,1\}^{c \cdot n}$ computable in NC^0. Then, there is no PTAS for Max3SAT.*

Proof: Let ε be a sufficiently small constant for which $\mathrm{H}_2(\varepsilon) \cdot c < 0.9$. Let $A_n \overset{\text{def}}{=} \mathrm{Com}(U_n, U_{m(n)})$ and $B_n \overset{\text{def}}{=} \mathrm{Com}(0^n, U_{m(n)})$. We show that A_n, B_n, $\delta(n) = 1 - 2^{-o(n)}$ and ε satisfy Assumption 6.4.1. Again, we focus on proving that the second item of the assumption holds.

Fix some $r \in \{0,1\}^{m(n)}$. There are at most $\sum_{i=0}^{\varepsilon cn} \binom{cn}{i} \leq 2^{\mathrm{H}_2(\varepsilon)cn} \leq 2^{0.9n}$ strings which are ε-close to $\mathrm{Com}(0^n, r)$. Hence, by the binding property, we have

$$\Pr[\mathrm{Com}(0^n, r) \text{ is } \varepsilon\text{-close to support}(\mathrm{Com}(U_n, U_{m(n)}))] \leq 2^{0.9n - n} = 2^{-0.1n}.$$

Thus,

$$\Pr_{r \leftarrow U_{m(n)}} [\textsc{Com}(0^n, r) \text{ is } \varepsilon\text{-close to support}(\textsc{Com}(U_n, U_{m(n)}))] \le 2^{-0.1n} < 1 - \delta(n),$$

where the last inequality holds for sufficiently large n. ■

Public-Key Encryption. Suppose we have an error-free public-key encryption scheme whose encryption algorithm is in NC^0 and whose information rate is constant (i.e., the ciphertext length is linear in the message length). Then, we can construct a (collection of) constant-rate NC^0 non-interactive commitments. (Set $\textsc{Com}_e(x, r) \stackrel{\text{def}}{=} E_e(x, r)$ where $E_e(x, r)$ is the encryption function which encrypts the message x using the key e and randomness r.) Hence, such a scheme also implies the inapproximability of Max3SAT.

6.5 A Construction of LPRG in NC^0

6.5.1 Overview

We start with an informal description of our construction. Consider the following distribution: *fix* a sparse matrix $M \in \{0,1\}^{m \times n}$ in which every row contains a constant number of ones, multiply it with a random n-bit vector x, and add a noise vector $e \in \{0,1\}^m$ which is uniformly distributed over all m-bits vectors whose Hamming weight is $\lceil \mu \cdot m \rceil$. (For concreteness, think of $m = 5n$ and $\mu = 0.1$.) That is, we consider the distribution $\hat{D}_\mu(M) \stackrel{\text{def}}{=} M \cdot x + e$, where all arithmetic is over $\mathbb{F}_2$.

Consider the distribution $\hat{D}_{\mu+m^{-1}}(M)$ which is similar to the previous distribution except that this time the noise vector is uniformly distributed over m-bit vectors whose weight is $(\mu+1/m)\cdot m = \mu m + 1$. Alekhnovich conjectured in [Ale03, Conjecture 1] that for a proper choice of M these distributions are computationally indistinguishable. He also showed that if indeed this is the case, then $\hat{D}_\mu(M)$ is pseudorandom; that is, $\hat{D}_\mu(M)$ is computationally indistinguishable from U_m. Since the distribution $\hat{D}_\mu(M)$ can be sampled (efficiently) by using roughly $n + \log \binom{m}{\mu \cdot m} \le n + m\mathrm{H}_2(\mu)$ random bits, it gives rise to a pseudorandom generator with linear stretch (when the parameters are chosen properly).

We would like to sample $\hat{D}_\mu(M)$ by an NC^0 function. Indeed, since the rows of M contains only a constant number of ones, we can easily compute the product Mx in NC^0 (recall that M itself is fixed). Unfortunately, we do not know how to sample the noise vector e by an NC^0 function. To solve this, we change the noise distribution. That is, we consider a slightly different distribution $D_\mu(M)$ in which each entry of the noise vector e is chosen to be 1 with probability μ (independently of other entries). We adopt Alekhnovich's conjecture to this setting; namely, we assume that $D_\mu(M)$ cannot be distinguished efficiently from $D_{\mu+m^{-1}}(M)$. (In fact, the new assumption is implied by the original one. See Section 6.5.5.) Similarly to the previous case, we show that under this assumption $D_\mu(M)$ is pseudorandom.

Now, whenever $\mu = 2^{-t}$ for some integer t, we can sample each bit of the noise vector by taking the product of t random bits. Hence, in this case $D_\mu(M)$ is samplable in NC^0 (as we think of μ as a constant). The problem is that our NC^0 procedure which samples $D_\mu(M)$ consumes more bits than it produces (i.e., it consumes $n + t \cdot m$ bits and produces m bits). Hence, we lose the stretch. To solve this, we note that most of the entropy of the seed was not used. Thus, we can gain more output bits by applying a randomness extractor to the seed. To be useful, this randomness extractor should be computable in NC^0. We construct such an extractor by relying on the construction of ε-biased generator in NC^0 of [MST03].

For ease of presentation, we describe our construction in a non-uniform way. We will later discuss a uniform variant of the construction.

6.5.2 The Assumption

Let $m = m(n)$ be an output length parameter where $m(n) > n$, let $\ell = \ell(n)$ be a locality parameter (typically a constant), and let $0 < \mu < 1/2$ be a noise parameter. Let $\mathcal{M}_{m,n,\ell}$ be the set of all $m \times n$ matrices over $\mathbb{F}_2$ in which each row contains at most ℓ ones. For a matrix $M \in \mathcal{M}_{m,n,\ell}$ denote by $D_\mu(M)$ the distribution of the random m-bit vector

$$Mx + e,$$

where $x \leftarrow U_n$ and $e \in \{0,1\}^m$ is a random error vector in which each entry is chosen to be 1 with probability μ (independently of other entries), and arithmetic is over $\mathbb{F}_2$. The following assumption is a close variant of a conjecture suggested by Alekhnovich in [Ale03, Conjecture 1]. [3]

Intractability Assumption 6.5.1 *For any $m(n) = O(n)$, and any constant $0 < \mu < 1/2$, there exists a positive integer ℓ, and an infinite family of matrices $\{M_n\}_{n\in\mathbb{N}}$, $M_n \in \mathcal{M}_{m(n),n,\ell}$, such that*

$$D_\mu(M_n) \overset{c}{\equiv} D_{\mu+m(n)^{-1}}(M_n)$$

(Note that since we consider non-uniform distinguishers, we can assume that M_n is public and is available to the distinguisher.)

Remark 6.5.2 Note that in Assumption 6.5.1 we do not require $\{M_n\}$ to be polynomial-time computable. We will later present a uniform construction based on the following version of Assumption 6.5.1. For any $m(n) = O(n)$, any constant $0 < \mu < 1/2$, and any infinite family of $m(n) \times n$ binary matrices $\{M_n\}_{n\in\mathbb{N}}$, if $\{M_n\}$ is expanding then $D_\mu(M_n) \overset{c}{\equiv} D_{\mu+m(n)^{-1}}(M_n)$. This assumption seems likely as argued by Alekhnovich [Ale03, Remark 1].

The following lemma shows that if the distribution $D_\mu(M_n)$ satisfies the above assumption then it is pseudorandom. (The proof is very similar to the proof of [Ale03, Theorem 3.1], and it is given here for completeness.)

Lemma 6.5.3 *For any polynomial $m(n)$ and constant $0 < \mu < 1/2$, and any infinite family, $\{M_n\}_{n\in\mathbb{N}}$, of $m(n) \times n$ matrices over $\mathbb{F}_2$, if $D_\mu(M_n) \overset{c}{\equiv} D_{\mu+m(n)^{-1}}(M_n)$, then $D_\mu(M_n) \overset{c}{\equiv} U_{m(n)}$.*

[3] Our assumption is essentially the same as Alekhnovich's. The main difference between the two assumptions is that the noise vector e in [Ale03] is a random vector of weight exactly $\lceil \mu m \rceil$, as opposed to our noise vector whose entries are chosen to be 1 independently with probability μ. In Section 6.5.5 we show that our assumption is implied by Alekhnovich's assumption. Intuitively, the implication follows from the fact that our noise vectors can be viewed as a convex combination of noise vectors of fixed weight. We do not know whether the converse implication holds. Indeed, a distribution D which can be described as a convex combination of distributions $D_1, \ldots, D_n$ may be pseudorandom even if each of the distributions D_i is not pseudorandom.

Proof: Let $m = m(n)$. Let r_n denote the distribution of an m-bit vector in which each entry is chosen to be 1 with probability c/m (independently of other entries) where c is the constant $1/(1-2\mu)$. As shown next, we can write

$$D_{\mu+m^{-1}}(M_n) \equiv D_\mu(M_n) + r_n. \tag{6.5.1}$$

To see this, let $e, e' \in \{0,1\}^m$ be noise vectors of rate $\mu, \mu + 1/m$ respectively. Then, to prove Eq. 6.5.1 it suffices to show that $e' \equiv e + r_n$. Indeed, the entries of $e + r_n$ are iid Bernoulli random variables whose success probability is

$$\mu \cdot (1 - (m(1-2\mu))^{-1}) + (1-\mu) \cdot (m(1-2\mu))^{-1} = \mu + m(n)^{-1}.$$

Now, by Eq. 6.5.1 and the lemma's hypothesis, we have

$$D_\mu(M_n) \overset{c}{\equiv} D_\mu(M_n) + r_n. \tag{6.5.2}$$

Let r_n^i be the distribution resulting from summing (over $\mathbb{F}_2^m$) i independent samples from r_n. Let $p(\cdot)$ be a polynomial. Then, by Fact 2.2.11, we get that

$$D_\mu(M_n) \overset{c}{\equiv} D_\mu(M_n) + r_n^{p(n)}. \tag{6.5.3}$$

Recall that r_n is a vector of iid Bernoulli random variables whose success probability is $\Theta(1/m)$. Hence, for some polynomial $p(\cdot)$ (e.g., $p(n) = nm$) it holds that

$$r_n^{p(n)} \overset{s}{\equiv} U_{m(n)}. \tag{6.5.4}$$

(To see this, note that $r_n^{p(n)}$ is a vector of iid Bernoulli random variables whose success probability is, by Fact 6.2.2, $1/2 \pm (1/2 - \Theta(1/m))^{p(n)} = 1/2 \pm \mathrm{neg}(n)$.) By combining Eq. 6.5.3 and 6.5.4, we have

$$D_\mu(M_n) \overset{c}{\equiv} D_\mu(M_n) + r_n^{p(n)} \overset{s}{\equiv} D_\mu(M_n) + U_{m(n)} \equiv U_{m(n)},$$

and the lemma follows. ■

By combining Assumption 6.5.1 and Lemma 6.5.3, we get the following.

Proposition 6.5.4 *Suppose that Assumption 6.5.1 holds. Then, for any $m(n) = O(n)$, and any constant $0 < \mu < 1/2$, there exists a constant $\ell \in \mathbb{N}$, and an infinite family of matrices $\{M_n\}_{n\in\mathbb{N}}$ where $M_n \in \mathcal{M}_{m(n),n,\ell}$ such that $D_\mu(M_n) \overset{c}{\equiv} U_{m(n)}$.*

Remark 6.5.5 If the restriction on the density of the matrices M_n is dropped, the above proposition can be based on the conjectured (average case) hardness of decoding a random linear code (cf., [BFKL94, GKL93]). In fact, under the latter assumption we have that $D_\mu(M_n) \overset{c}{\equiv} U_{m(n)}$ for *most* choices of M_n's.

6.5.3 The Construction

From here on, we let $\mu = 2^{-t}$ for some $t \in \mathbb{N}$. Then, we can sample each bit of the error vector e by taking the product of t independent random bits. This naturally gives rise to an NC^0 function whose output distribution is pseudorandom, namely,

$$f_n(x, y) = M_n x + E(y)$$

where

$$x \in \{0,1\}^n, \qquad y \in \{0,1\}^{t \cdot m(n)}, \qquad E(y) = \left(\prod_{j=1}^{t} y_{t\cdot(i-1)+j} \right)_{i=1}^{m(n)}. \tag{6.5.5}$$

Since $f_n(U_n, U_{t\cdot m(n)}) \equiv D_\mu(M_n)$, the distribution $f_n(U_n, U_{t\cdot m(n)})$ is pseudorandom under Assumption 6.5.1 (when the parameters are chosen appropriately). Moreover, the locality of f_n is $\ell + t = O(1)$. However, f_n is not a pseudorandom generator as it uses $n + t \cdot m(n)$ input bits while it outputs only $m(n)$ bits. To overcome this obstacle, we note that most of the entropy of y was not "used". Indeed, we use the $t \cdot m(n)$ random bits of y to sample the distribution $E(y)$ whose entropy is only $m(n) \cdot \mathrm{H}_2(2^{-t}) < (t+2) \cdot 2^{-t} \cdot m(n)$. Hence, we can apply an *extractor* to regain the lost entropy. Of course, in order to get a PRG in NC^0 the extractor should also be computed in NC^0. Moreover, to get a linear stretch we should extract almost all of the $t \cdot m(n)$ random bits from y while investing less than m additional random bits. In the following, we show that such extractors can be implemented by using *ε-biased generators.*

First, we show that the distribution of y given $E(y)$ contains (with high probability) a lot of entropy. In the following we let $m = m(n)$.

Lemma 6.5.6 *Let $y \leftarrow U_{t\cdot m}$ and $E(y)$ be defined as in Eq. 6.5.5. Denote by $[y|E(y)]$ the distribution of y given the outcome of $E(y)$. Then, except with probability $e^{-(2^{-t}m)/3}$ over the choice of y, it holds that*

$$\mathrm{H}_\infty([y|E(y)]) \geq (1 - \delta(t)) \cdot tm, \tag{6.5.6}$$

where $\delta(t) = 2^{-\Omega(t)}$.

Proof: We view $E(y)$ as a sequence of m independent Bernoulli trials, each with a probability 2^{-t} of success. Recall that y is composed of m blocks of length t, and that the i-th bit of $E(y)$ equals the product of the bits in the i-th block of y. Hence, whenever $E(y)_i = 1$ all the bits of the i-th block of y equal to 1, and when $E(y)_i = 0$ the i-th block of y is uniformly distributed over $\{0,1\}^t \setminus \{1^t\}$. Consider the case in which at most $2 \cdot 2^{-t}m$ components of $E(y)$ are ones. By a Chernoff bound, the probability of this event is at least $1 - e^{-(2^{-t}m)/3}$. In this case, y is uniformly distributed over a set of size at least $(2^t - 1)^{(1-2^{-t+1})m}$. Hence, conditioning on the event that at most $2 \cdot 2^{-t}m$ components of $E(y)$ are ones, the min-entropy of $[y|E(y)]$ is at least $m(1 - 2^{-t+1})\log(2^t - 1) \geq tm(1 - \delta(t))$, for $\delta(t) = 2^{-\Omega(t)}$. ■

ε-biased generators can be used to extract random bits from distributions that contain sufficient randomness. Extractors based on ε-biased generators were previously used in [DS05].

Lemma 6.5.7 ([DS05, Lemma 4]) *Let $g : \{0,1\}^n \to \{0,1\}^s$ be an ε-biased generator, and let X_s be a random variable taking values in $\{0,1\}^s$ whose min-entropy is at least $(1-\delta)\cdot s$, for some $\delta \geq 0$. Then,*

$$\mathrm{SD}((g(U_n) + X_s), U_s) \leq \varepsilon \cdot 2^{\delta \cdot s/2 - 1/2} \ ,$$

where vector addition is taken over $\mathbb{F}_2$.

The above lemma follows directly by analyzing the affect of a random step over a Cayley graph whose generator set is an ε-biased set (cf. [GW97, Lemma 2.3] and [NN93, AR94]).

Recently, Mossel et al. [MST03] constructed an ε-biased generator in nonuniform-NC^0_5 with an arbitrary linear stretch and exponentially small bias.

Lemma 6.5.8 ([MST03, Thm. 14]) *For every constant c, there exists an ε-biased generator $g : \{0,1\}^n \to \{0,1\}^{cn}$ in nonuniform-NC^0_5 whose bias is at most $2^{-bn/c^4}$ (where $b > 0$ is some universal constant that does not depend on c).*

In Section 6.5.4 we provide an uniform version of the above lemma in which the bias is only $2^{-n/\mathrm{polylog}(c)}$. The price we pay is in the locality which grows polylogarithmically with the stretch constant c. (See Theorem 6.5.12.)

We can now describe our LPRG.

Construction 6.5.9 *Let t and ℓ be positive integers, and $c, k > 1$ be real numbers that will effect the stretch factor. Let $m = kn$ and let $\{M_n \in \mathcal{M}_{n,m,\ell}\}$ be an infinite family of matrices. Let $g : \{0,1\}^{tm/c} \to \{0,1\}^{tm}$ be the ε-biased generator promised by Lemma 6.5.8. We define the function*

$$G_n(x, y, r) = (M_n x + E(y), g(r) + y),$$

where $x \in \{0,1\}^n, y \in \{0,1\}^{t\cdot m}, r \in \{0,1\}^{t\cdot m/c}, E(y) = \left(\prod_{j=1}^{t} y_{t\cdot(i-1)+j}\right)_{i=1}^{m}$. Thus, $G_n : \{0,1\}^{n+tm+\frac{tm}{c}} \to \{0,1\}^{m+tm}$.

Observe that G_n is in nonuniform-NC^0. We show that if the parameters are chosen properly then G_n is an LPRG.

Lemma 6.5.10 *Under Assumption 6.5.1, there exist constants $t, \ell \in \mathbb{N}$, constants $c, k > 1$, and a family of matrices $\{M_n \in \mathcal{M}_{n,m,\ell}\}$ such that the function G_n defined in Construction 6.5.9 is an LPRG.*

Proof: Let $k > 1$ be some arbitrary constant and $m = m(n) = kn$. Let c and t be constants such that:

$$c = 2t/(1 - 1/k)$$

and

$$\Delta \overset{\text{def}}{=} t\left(\frac{b}{c^5} - \delta(t)\right) > 0, \tag{6.5.7}$$

where $\delta(\cdot)$ is the negligible function from Eq. 6.5.6 and b is the bias constant of Lemma 6.5.8. Such constants c and t do exist since $\delta(t) = 2^{-\Omega(t)}$ while $b/c^5 = \Theta(1/t^5)$. Let $\ell \in \mathbb{N}$ be a constant and $\{M_n \in \mathcal{M}_{n,m,\ell}\}$ be an infinite family of matrices satisfying Assumption 6.5.1.

First, we show that G_n has linear stretch. The input length of G_n is $n + tm + tm/c = (tk + k/2 + 1/2) \cdot n$. The output length is $(t + 1) \cdot m = (tk + k) \cdot n$. Hence, since $k > 1$, the function G_n has a linear stretch.

Let x, y and r be uniformly distributed over $\{0,1\}^n, \{0,1\}^{t \cdot m}$ and $\{0,1\}^{t \cdot m/c}$ respectively. We prove that the distribution $G_{M_n}(x, y, r)$ is pseudorandom. By Fact 2.2.4 and Lemmas 6.5.6, 6.5.7 and 6.5.8 it holds that

$$\begin{aligned}
\mathrm{SD}((E(y), y + g(r)), (E(y), U_{t \cdot m})) &\leq e^{-(2^{-t}m)/3} + 2^{-b \cdot (tm/c)/c^4} \cdot 2^{tm \cdot \delta(t)/2 - 1/2} \\
&\leq e^{-(2^{-t}m)/3} + 2^{(-b/c^5 + \delta(t)) \cdot tm} \\
&\leq e^{-(2^{-t}m)/3} + 2^{-\Delta m} = \mathrm{neg}(m) = \mathrm{neg}(n),
\end{aligned}$$

where the last inequality is due to Eq. 6.5.7. Therefore, by Fact 2.2.3 and Proposition 6.5.4, we get that

$$(M_n x + E(y), g(r) + y) \overset{\mathrm{s}}{\equiv} (M_n x + E(y), U_{t \cdot m}) \equiv (D_{2^{-t}}(M_n), U_{t \cdot m}) \overset{\mathrm{c}}{\equiv} (U_m, U_{t \cdot m}),$$

and the lemma follows. ■

By the above lemma we get a construction of LPRG in nonuniform-NC^0 from Assumption 6.5.1. Moreover, by combining the above with Theorem 4.5.7 we have:

Theorem 6.5.11 *Under Assumption 6.5.1, there exists an LPRG in nonuniform-NC^0_4.*

Mossel et al. [MST03] showed that a PRG in nonuniform-NC^0_4 cannot achieve a superlinear stretch. Hence, Theorem 6.5.11 is essentially optimal with respect to stretch.

Remarks on Theorem 6.5.11.

1. (Uniformity) Our construction uses two non-uniform advices: (1) a family of good ε-biased generators in NC^0 as in Lemma 6.5.8; and (2) a family of matrices $\{M_n\}$ satisfying Assumption 6.5.1. In Section 6.5.4 we eliminate the use of the first advice by proving a uniform version of Lemma 6.5.8. We can also eliminate the second advice and construct an LPRG in *uniform* NC^0_4 by using an explicit variant of Assumption 6.5.1. In particular, we follow Alekhnovich (cf. [Ale03, Remark 1]) and conjecture that any family of matrices $\{M_n\}$ that represent graphs with good expansion satisfies Assumption 6.5.1. Hence, our construction can be implemented by using an explicit family of asymmetric constant-degree bipartite expanders such as the one given in [CRVW02, Theorem 7.1].

2. (PRG with constant input locality) A variant of our construction also gives a PRG G in NC^0 in which each input bit affects a constant number of output bits. Namely, G enjoys from constant output locality and constant *input* locality at the same time. This can be done by instantiating Construction 6.5.9 with a family of matrices $\{M_n\}$ in which each row and each *column* contain a constant number of 1's, and in addition, employing an ε-biased generator with constant input locality and constant output locality. It turns out that the instantiation proposed in the previous item (which is based on [CRVW02, Theorem 7.1] and Theorem 6.5.12) satisfies these conditions. Thus, assuming that Assumption 6.5.1 holds whenever $\{M_n\}$ is expanding, we get a (uniform) LPRG with constant output locality and constant input locality. In Chapter 7, we will construct a (collection) of PRGs with better (optimal) input and output locality under a weaker assumption (namely, the intractability of decoding random linear code). However, the construction of Chapter 7 is limited to *sublinear* stretch.

3. (The stretch of the construction) Our techniques do not yield a PRG with *superlinear* stretch in NC^0. To see this, consider a variant of Assumption 6.5.1 in which we allow $m(n)$ to be superlinear. If we let $\mu(n)$ to be a constant, then, by information-theoretic arguments, we need $\Omega(m(n))$ random bits to sample the noise vector (i.e., the entropy of the noise vector is $\Omega(m(n))$), and so we get only linear stretch. On the other hand, if we set $\mu(n)$ to be subconstant, then the noise distribution cannot be sampled in NC^0 (as any bit of an NC^0-samplable distribution depends on a constant number of random bits). This problem can be bypassed by extending Assumption 6.5.1 to alternative noise models in which the noise is not independently and identically distributed. However, it is not clear how such a modification affects the hardness assumption. (Also note that we do not know how to reduce the locality of a superlinear PRG in NC^0 while preserving its superlinear stretch. In particular, applying the transformations of Chapter 4 to such a PRG, will result in a *linear* PRG with locality 4.)

6.5.4 ε-Biased Generators in Uniform NC^0

In [MST03, Theorem 14], Mossel et al. constructed an ε-biased generator in nonuniform-NC^0_5 with an arbitrary linear stretch cn and bias $\varepsilon = 2^{-\Omega(n/c^4)}$.[4] We generalize their construction and provide a complementary result which gives a better tradeoff between the bias and stretch and allows a uniform implementation. However, the locality of our construction grows with the stretch constant.

Theorem 6.5.12 *For every constant c, there exist an ε-biased generator $g : \{0,1\}^n \to \{0,1\}^{cn}$ in* uniform NC^0 *whose bias is $\varepsilon = 2^{-n/\mathrm{polylog}(c)}$ and its output locality is $\ell = \mathrm{polylog}(c)$. Moreover, the input locality of g is constant (which depends on c).*

As in [MST03], our generator is obtained by XORing the outputs of two functions: a generator $g^{(s)}$ which is robust against linear functions that involve small number of output bits ("small tests") and a generator $g^{(l)}$ which is robust against linear functions that involve large number of output bits ("large tests"). More precisely, for a random variable $X = (X_1, \ldots, X_n)$ ranging over $\{0,1\}^n$, a set $S \subseteq \{1, \ldots, n\}$, and an integer $0 < k \leq n$, we define

$$\begin{aligned}
\mathrm{bias}_S(X) &\stackrel{\mathrm{def}}{=} \left| \Pr[\bigoplus_{i \in S} X_i = 0] - \frac{1}{2} \right|, \\
\mathrm{bias}_k(X) &\stackrel{\mathrm{def}}{=} \max_{S \subseteq \{1,\ldots,n\}, |S|=k} \mathrm{bias}_S(X), \\
\mathrm{bias}(X) &\stackrel{\mathrm{def}}{=} \max_{0<k\leq n} \mathrm{bias}_k(X) = \max_{S \subseteq \{1,\ldots,n\}, S \neq \emptyset} \mathrm{bias}_S(X) \ .
\end{aligned}$$

Then, we prove Theorem 6.5.12 by using the following two lemmas (whose proofs is postponed to Sections 6.5.4, 6.5.4):

Lemma 6.5.13 (Generator against small tests) *For every constant c, there exist a function $g^{(s)} : \{0,1\}^n \to \{0,1\}^{cn}$ in* uniform $\mathrm{NC}^0_{\mathrm{polylog}(c)}$ *such that for sufficiently large n's and every $0 < k \leq \Omega(n/\mathrm{polylog}(c))$, we have $\mathrm{bias}_k(g^{(s)}(U_n)) = 0$. Moreover, the input locality of $g^{(s)}$ is constant (which depends on c).*

[4] In fact, cn can be slightly super-linear.

Lemma 6.5.14 (Generator against large tests) *For every constant c, there exist a function $g^{(l)} : \{0,1\}^n \to \{0,1\}^{cn}$ in* uniform $\mathrm{NC}^0_{O(\log(c))}$ *such that for sufficiently large n's and every $k \in \{1, \dots, cn\}$, we have* $\mathrm{bias}_k(g^{(l)}(U_n)) \le 2^{-k/5}$. *Moreover, the input locality of $g^{(s)}$ is constant (which depends on c).*

Given these two lemmas we can prove Theorem 6.5.12.
Proof of Theorem 6.5.12: Let c be a constant. Let $g^{(s)} : \{0,1\}^n \to \{0,1\}^{2cn}$ and $g^{(l)} : \{0,1\}^n \to \{0,1\}^{2cn}$ be the generators promised by Lemmas 6.5.13, 6.5.14 (instantiate with the constant $2c$). Then, the function $g(x,y) = g^{(s)}(x) \oplus g^{(l)}(y)$ satisfies Theorem 6.5.12. To see this, observe that for any *independent* random variables X and Y and any non-uniform statistical test T, the success probability of T on the random variable $X \oplus Y$ is not larger than its success probability on X (or Y). ■

Proof of Lemma 6.5.13

Let M be an $m \times n$ matrix over $\mathbb{F}_2$ such that every subset of k rows of M are linearly independent. Then, it is well known that the function $f : \{0,1\}^n \to \{0,1\}^m$ that maps x into $M \cdot x$ is a k-wise independent generator (cf. [ABI86]). That is, for every $0 < j \le k$, we have $\mathrm{bias}_j(f(U_n)) = 0$. If each row of M contains at most ℓ ones then the function f is in nonuniform-NC^0_ℓ. It turns out that there exists a (uniform) family of such matrices whose parameters match the parameters of Lemma 6.5.13. Specifically, we use the following result which is a corollary of [CRVW02, Theorem 7.1].

Lemma 6.5.15 ([CRVW02]) *For every constant c there exists a family of matrices $\{M_n\}_{n \in \mathbb{N}}$ such that*

- *M_n is an $cn \times n$ matrix over $\mathbb{F}_2$.*
- *Every row of M_n has at most* $\mathrm{polylog}(c)$ *ones.*
- *Every column of M_n has at most $d(c)$ ones for some function $d(\cdot)$.*
- *Every subset of $k = \Omega(n/\mathrm{polylog}(c))$ rows of M_n are linearly independent.*
- *M_n can be constructed in time* $\mathrm{poly}(n)$.

Hence, the generator for small tests can be defined as $g^{(s)}(x) = M_n \cdot x$.

Proof of Lemma 6.5.14

We will need the following standard claim that can be proved via the probabilistic method (see [Gol01b, Lecture 8, Prop. 2.1]).

Claim 6.5.16 *For sufficiently large n, there exists an ε-biased generator $f : \{0,1\}^n \to \{0,1\}^{2^{n/2}}$ whose bias is $\varepsilon = 2^{-n/4}$.*

We can now prove Lemma 6.5.14. Let c be the desired stretch constant. Let $\ell = 4 \log c$. Let $m = 2^{\ell/2}$ and $f : \{0,1\}^\ell \to \{0,1\}^m$ be an ε-biased generator whose bias is $\varepsilon = 2^{-\ell/4}$ as promised by Claim 6.5.16. (Since c is a constant, such f can be found by using exhaustive search.) Our generator will partition its n-bit input x into $b = \lfloor n/\ell \rfloor$ blocks $x^{(1)}, \dots, x^{(b)}$ of length ℓ each. Then, the generator will apply f to each block separately, and concatenate the result. Namely, $g^{(l)}(x) \overset{\text{def}}{=} (f(x^{(1)}), \dots, f(x^{(b)}))$. The output locality of $g^{(l)}$ is ℓ, its input locality is $m = 2^{\ell/2} = c^2$ and its output length is $bm = \left\lfloor \frac{c^2 n}{4 \log c} \right\rfloor$ which is larger than cn for sufficiently large c.

We now analyze the bias of $g^{(l)}$. To simplify notation, we index the outputs of $g^{(l)}$ by pairs (j, i) and let $g^{(l)}_{j,i}(x) = f_i(x^{(j)})$ (where $1 \le j \le b$, $1 \le i \le m$ and $f_i(x)$ denotes the i-th output bit of $f(x)$). Let $S \subseteq \{1, \dots b\} \times \{1, \dots m\}$ be a linear test of cardinality k. Let S_j be the restriction of S to the indices of the j-th block, i.e., $S_j = \{i : (j, i) \in S\}$. Then, $S_1, \dots, S_b$ is a partition of S. Let $T = \{i : S_i \neq \emptyset\} \subseteq \{1, \dots, b\}$. Hence, for $x \leftarrow U_n$, we have

$$\text{bias}_S(g^{(l)}(x)) \quad = \quad \text{bias}\left(\bigoplus_{j \in T} \bigoplus_{i \in S_j} f_i(x^{(j)}) \right).$$

Since f is an ε-biased generator, for each $j \in T$ we have that $\text{bias}(\bigoplus_{i \in S_j} f_i(x^{(j)})) \le \varepsilon$. Since $g^{(l)}(x)$ is partitioned into blocks of length ℓ, the test S contains output bits coming from at least k/ℓ different blocks and so $|T| \ge k/\ell$. Thus we can use Fact 6.2.2 to upper bound $\text{bias}_S(g^{(l)}(x))$ by

$$\frac{1}{2}(2\varepsilon)^{k/\ell} \le \frac{1}{2}(2^{-\ell/4+1})^{k/\ell} \le \frac{1}{2}(2^{-\ell/5})^{k/\ell} \le 2^{-k/5},$$

as required. ■

6.5.5 Alekhnovich's Assumption Implies Assumption 6.5.1

We show that Alekhnovich's Assumption [Ale03, Conjecture 2, Remark 1] implies Assumption 6.5.1. The main difference between the two assumptions is that the noise vector e in [Ale03] is a random vector of weight exactly $\lceil \mu m \rceil$, as opposed to our noise vector whose entries are chosen to be 1 independently with probability μ. The implication follows from the fact that our noise vectors can be viewed as a convex combination of noise vectors of fixed weight. We give the details below.

Recall that for an $m \times n$ matrix M we let $\hat{D}_\mu(M)$ denote the distribution of $M \cdot x + e$, where x is a random n-bit vector and e is a noise vector which is uniformly distributed over all m-bits vectors whose Hamming weight is $\mu \cdot m$. The distribution $D_\mu(M) \overset{\text{def}}{=} M \cdot x + e$ is analogous to $\hat{D}_\mu(M)$, except that each entry of the noise vector e is chosen to be 1 with probability μ (independently of other entries).

Intractability Assumption 6.5.17 ***(Alekhnovich's Assumption)*** *For any $m(n) = O(n)$, there exists an infinite family of matrices $\{M_n\}_{n \in \mathbb{N}}$, $M_n \in \mathcal{M}_{m(n),n,3}$, such that for any constant $0 < \mu_0 < 1/2$, and function $\mu(n)$ that satisfies $\mu_0 < \mu(n) < 1/2$ for every n, it holds that*

$$\hat{D}_{\mu(n)}(M_n) \overset{c}{\equiv} \hat{D}_{\mu(n)+m(n)^{-1}}(M_n).$$

Fix a matrix family $\{M_n\}_{n \in \mathbb{N}}$ of size $m(n) \times n$ where $m(n)$ is an integer valued function. We will prove that Assumption 6.5.17 instantiated with the family $\{M_n\}_{n \in \mathbb{N}}$ implies Assumption 6.5.1 instantiated with the same family of matrices. To do this we use the following two intermediate assumptions.

Intractability Assumption 6.5.18 *For any constant* $0 < \mu_0 < 1/2$*, and function* $\mu(n)$ *that satisfies* $\mu_0 < \mu(n) < 1/2$ *for all* n*'s,* $\hat{D}_{\mu(n)}(M_n) \overset{c}{\equiv} U_{m(n)}$*.*

Intractability Assumption 6.5.19 *For any constant* $0 < \mu < 1/2$*, we have* $D_\mu(M_n) \overset{c}{\equiv} U_{m(n)}$*.*

In [Ale03, Thm. 3.1] it is shown that Assumption 6.5.17 implies Assumption 6.5.18. Hence to prove that Assumption 6.5.17 implies Assumption 6.5.1 it suffices to show that: (1) Assumption 6.5.18 implies Assumption 6.5.19; and (2) Assumption 6.5.19 implies Assumption 6.5.1.

Lemma 6.5.20 *Assumption 6.5.18 implies Assumption 6.5.19.*

Proof: Suppose that Assumption 6.5.19 does not hold. Then, for some constant $0 < \mu < 1/2$, the distribution $D_\mu(M_n)$ is not pseudorandom. That is, there exists a polynomial-size circuit family $\{A_n\}$ and a polynomial $q(\cdot)$ such that

$$\Pr[A_n(D_\mu(M_n)) = 1] - \Pr[A_n(U_{m(n)}) = 1] > 1/q(n), \tag{6.5.8}$$

for infinitely many n's. We will show that, for some constant $0 < \hat{\mu}_0 < 1/2$, and function $\hat{\mu}(n)$ that satisfies $\hat{\mu}_0 < \hat{\mu}(n) < 1/2$, Assumption 6.5.18 is violated. Namely, $\Pr[A_n(\hat{D}_{\hat{\mu}(n)}(M_n)) = 1] - \Pr[A_n(U_{m(n)}) = 1] > 1/q'(n)$ for some polynomial $q'(\cdot)$ and infinitely many n's.

Fix some n for which Eq. 6.5.8 holds, and let $m = m(n)$. Let $p \overset{\text{def}}{=} \Pr[A_n(D_\mu(M_n)) = 1]$ and $p(k) \overset{\text{def}}{=} \Pr[A_n(\hat{D}_{k/m}(M_n)) = 1]$ for $0 \le k \le m$. Let $e \in \{0,1\}^m$ be a random error vector in which each entry is chosen to be 1 with probability μ (independently of other entries) and let $t(k)$ be the probability that e contains exactly k ones. We can think of the distribution of e as the outcome of the following process: first choose $0 \le k \le m$ with probability $t(k)$, then choose a random noise vector of weight k. Hence, we can write,

$$p = \sum_{k=0}^{m} p(k) \cdot t(k).$$

Let $\varepsilon > 0$ be a constant for which $\mu \cdot \varepsilon < 1/2$. Then, by a Chernoff bound, it holds that

$$\sum_{k<(1-\varepsilon)\cdot \mu m} t(k) + \sum_{k>(1+\varepsilon)\cdot \mu m} t(k) = \Pr\left[\left|\sum_{i=1}^{m} e_i - \mu m\right| > \varepsilon \cdot \mu m\right] < 2e^{-\varepsilon^2 \mu m/3}.$$

Hence,

$$\sum_{(1-\varepsilon)\cdot\mu m \le k \le (1+\varepsilon)\cdot \mu m} p(k) \cdot t(k) > p - 2e^{-\varepsilon^2 \mu m/3}.$$

Thus, by an averaging argument, there exists some $(1-\varepsilon)\cdot \mu m \le k \le (1-\varepsilon)\cdot \mu m$ for which

$$p(k) > p - 2e^{-\varepsilon^2 \mu m/3}.$$

Let $\hat{\mu}(n)$ be k/m and let $\hat{\mu}_0$ be the constant $(1-\varepsilon)\cdot \mu m/2$. Then, by Eq. 6.5.8, we have

$$\Pr[A_n(\hat{D}_{\hat{\mu}(n)}(M_n)) = 1] - \Pr[A_n(U_{m(n)}) = 1] > 1/q(n) - 2e^{-\varepsilon^2 \mu m/3} > 1/q'(n),$$

where $q'(\cdot)$ is a polynomial. This completes the proof since $\hat{\mu}_0 < \hat{\mu}(n) < 1/2$ for every n. ■

It is left to prove the following lemma.

Lemma 6.5.21 *If Assumption 6.5.19 holds then Assumption 6.5.1 also holds with respect to $\{M_n\}_{n\in\mathbb{N}}$.*

Proof: As shown in the proof of Lemma 6.5.3 we can write $D_{\mu+m^{-1}}(M_n) \equiv D_\mu(M_n) + r_n$, where r_n denotes the distribution of an m-bit vector in which each entry is chosen to be 1 with probability c/m (independently of other entries) for some constant c. Hence, by two invocations of Assumption 6.5.19, we have

$$D_{\mu+m^{-1}}(M_n) \equiv D_\mu(M_n) + r_n \overset{\text{c}}{\equiv} U_{m(n)} + r_n \equiv U_{m(n)} \overset{\text{c}}{\equiv} D_\mu(M_n).$$

■

6.6 The Necessity of Expansion

As pointed out in Section 6.5, our construction of LPRG makes use of expander graphs. This is also the case in several constructions of "hard functions" with low locality (e.g., [Gol00, MST03, Ale03]). We argue that this is not coincidental, at least in the case of PRGs. Namely, we show that the input-output graph of any LPRG in NC^0 enjoys some expansion property. (In fact, this holds even in the case of ε-biased generators.) Then, we use known lower bounds for expander graphs to rule out the possibility of exponentially strong PRG with superlinear stretch in NC^0. These results are discussed from a wider perspective in Section 6.6.2. We start with the technical results.

6.6.1 Actual Results

For a function $G : \{0,1\}^n \to \{0,1\}^s$, we define the input-output graph $H_G = ((\mathrm{Out} = [s], \mathrm{In} = [n]), E)$ to be the bipartite graph whose edges correspond to the input-output dependencies in G; that is, (i,j) is an edge if and only if the i-th output bit of G depends on the j-th input bit. When G is a function family, H_G denotes a graph family.

Proposition 6.6.1 *Let $G : \{0,1\}^n \to \{0,1\}^{s(n)}$ be a PRG. Then, the graph (family) $H_G = ((\mathrm{Out} = [s(n)], \mathrm{In} = [n]), E)$ enjoys the following output expansion property: for every constant c and sufficiently large n, every set of output vertices $T \subseteq \mathrm{Out}$ whose size is at most $c\log n$ touches at least $|T|$ input vertices.*

Proof: Assume towards a contradiction that there exists a small set T of output vertices that touches less than $|T|$ input vertices. Let $G_T(\cdot)$ be the restriction of G to the output bits of T. Then, the function $G_T(\cdot)$ cannot be onto as it depends on less than $|T|$ input bits. Therefore, there exists a string $z \in \{0,1\}^{|T|}$ such that $\Pr[G_T(U_n) = z] = 0$. Hence, a (non-uniform) distinguisher which given $y \in \{0,1\}^{s(n)}$ checks whether $y_T = z$, distinguishes between $G(U_n)$ and $U_{s(n)}$ with advantage $2^{-c\log n} = 1/n^c$, in contradiction to the pseudorandomness of G. ■

More generally, if G is ε-hard (i.e., cannot be broken by any efficient adversary with advantage ε), then every set of $t \leq \log(1/\varepsilon)$ output vertices touches at least t input vertices. This claim also extends to the case of ε-biased generators.

Proposition 6.6.2 *Let $G : \{0,1\}^n \to \{0,1\}^s$ be an ε-biased generator. Then, every set of $t \leq \log(1/\varepsilon)$ output vertices in H_G touches at least t input vertices.*

Proof: Assume towards a contradiction that there exists a set T of output vertices of size $t \leq \log(1/\varepsilon)$ that touches less than t input vertices. Then $G_T(U_n) \not\equiv U_t$. Therefore, there exists a linear function $L : \mathbb{F}_2^t \to \mathbb{F}_2$ that distinguishes between $G_T(U_n)$ and U_t. Namely, $|\Pr[L(G_T(U_n)) = 1] - \Pr[L(U_t) = 1]| \neq 0$. Since the distribution $G_T(U_n)$ is sampled by less than t random bits, the distinguishing advantage of L is larger than $2^{-t} \geq \varepsilon$, and so G is not ε-biased in contradiction to the hypothesis. ■

The above propositions show that when G is an ε-hard PRG (or even ε-biased generator), the bipartite graph $H_G = ((\mathrm{Out} = [s(n)], \mathrm{In} = [n]), E)$ enjoys some output expansion property. Radhakrishnan and Ta-Shma [RTS00] obtained some lower bounds for such graphs.

Proposition 6.6.3 ([RTS00, Theorem 1.5]) *Let $H = ((V_1 = [s], V_2 = [n]), E)$ be a bipartite graph in which every set $S \subseteq V_1$ of cardinality k touches at least m vertices from V_2. Then, the average degree of V_1 is at least $\Omega\left(\frac{\log(s/k)}{\log(m/n)}\right)$*

By combining this lower bound with the previous propositions we derive the following limitation on the strength of PRGs with superlinear stretch in NC^0.

Corollary 6.6.4 *Let $G : \{0,1\}^n \to \{0,1\}^s$ be a 2^{-t}-hard PRG (or 2^{-t}-biased generator). Then, the locality of G is at least $\Omega\left(\frac{\log(s/t)}{\log(n/t)}\right)$. In particular, there is no $2^{-\Omega(n)}$-hard PRG, or even a $2^{-\Omega(n)}$-biased generator, with superlinear stretch in* NC^0.

6.6.2 Discussion

To put the above results in context, some background on unbalanced bipartite expanders is needed. Consider a bipartite graph $H = ((\mathrm{Out} = [s], \mathrm{In} = [n]), E)$ in which each of the output vertices is connected to at most d inputs. Recall that H is a (K, α)-expander if every set of output vertices S of size smaller than K has at least $\alpha \cdot |S|$ input neighbors. We say that the expander is unbalanced if $s > n$. Unbalanced expanders have had numerous applications in computer science (see details and references in [CRVW02]). Today, there are only two such constructions [TSUZ01, CRVW02]. Ta-Shma et al. [TSUZ01] considered the highly unbalanced case in which $n < o(s)$. They constructed a (K, α)-expander with degree $d = \mathrm{polylog}(s)$, expansion threshold $K < s^{\varepsilon}$ and almost optimal expansion factor $\alpha = (1 - \delta)d$, where $\delta > 0$ is an arbitrary constant. Capalbo et al. [CRVW02] present a construction for the setting in which n is an (arbitrary) constant fraction of s (i.e., $s = n + \Theta(n)$). They construct a (K, α)-expander with (nearly) optimal parameters; Namely, the degree d of the graph is constant, and its expansion parameters are $K = \Omega(s)$ and $\alpha = (1 - \delta)d$, where $\delta > 0$ is an arbitrary constant.

In Section 6.6.1 we showed that if $G : \{0,1\}^n \to \{0,1\}^s$ is a PRG then its input-output graph $H_G = ((\mathrm{Out} = [s], \mathrm{In} = [n]), E)$ is an $(\omega(\log n), 1)$-expander. This property is trivial to satisfy when the output degree of H_G is unbounded (as in standard constructions of PRGs in which every output bit depends on all the input bits). It is also easy to construct such a graph with constant output degree when $s(n)$ is not much larger than n (as in the NC^0 constructions of Chapter 4.

To see this, consider the following bipartite graph. First, let $C = ((O, I), D)$ be a bipartite graph over $[2n+1]$ whose output vertices are the odd integers, its input vertices are the even integers, and its edges correspond to pairs of consecutive integers, i.e., $O = \{1, 3, \ldots, 2n+1\}$, $I = \{2, 4, \ldots, 2n\}$, and $D = \{(1,2), (2,3), \ldots, (2n, 2n+1)\}$. That is, C is a chain of length $2n + 1$. Let $m > n$. Take

m disjoint copies of C, and let O_i (resp. I_i) be the set of output (resp. input) vertices of the i-th copy. In addition, add n input vertices $I_0 = [n]$ and match them to the first n vertices of each of the output clusters (i.e., connect the j-th vertex of I_0 to the vertex $2j-1$ of each O_i). Let $H = ((\mathrm{Out} = O_1 \cup \cdots \cup O_m, \mathrm{In} = I_1 \cup \cdots \cup I_m \cup I_0), E)$. (See Figure 6.6.1.) Clearly, H has $m(n+1)$ output vertices, $mn+n$ input vertices, and each output vertex is connected to at most 3 inputs. It is not hard to verify that H is $(n^2, 1)$-expanding. However, the number of outputs is only slightly larger than the number of inputs; i.e., $|\mathrm{Out}| - |\mathrm{In}| = m - n < m$ which is sublinear in $|\mathrm{In}|$ when n is non-constant.

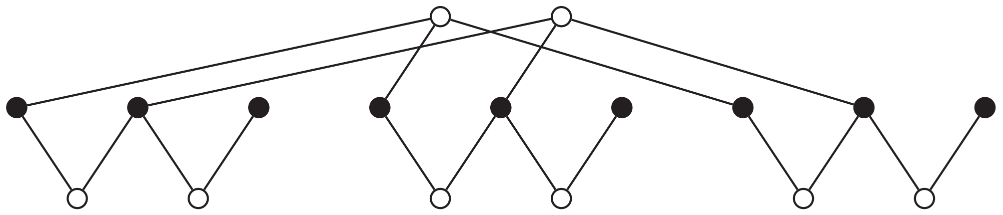

Figure 6.6.1: The graph H with $n = 2$ and $m = 3$. Black circles denote output vertices while empty circles denote input vertices.

However, when the locality of the pseudorandom generator G is constant and the stretch is linear, H_G is a sparse bipartite graph having n input vertices, $s(n) = n + \Omega(n)$ output vertices, and a constant output degree. It seems that it is not trivial to explicitly construct such a graph that achieves $(\omega(\log n), 1)$-expansion. (Indeed, the construction of [CRVW02] gives similar graphs whose expansion is even stronger, but this construction is highly non-trivial.) Hence, any construction of LPRG in NC^0 defines a non-trivial combinatorial structure. In particular, one cannot hope that "simple" *deterministic* transformations, such as those given in Chapter 4, will yield LPRGs in NC^0.

Note that an exponentially strong PRG (or exponentially strong ε-biased generator) with linear stretch gives an $(\Omega(n), 1)$-expander graph whose output size grows linearly with its input size. Indeed, the exponentially strong ε-biased generator of [MST03] is based on a similar (but slightly stronger) unbalanced expander. The above argument shows that such an ingredient is necessary.

Chapter 7

Cryptography with Constant Input Locality

Summary: So far we studied the possibility of implementing cryptographic tasks by functions with constant *output* locality. In this chapter we turn to the dual question of constant *input* locality. In particular, we ask: Which cryptographic primitives (if any) can be realized by functions with constant input locality, namely functions in which every bit of the *input* influences only a constant number of bits of the output? We (almost) characterize what cryptographic tasks can be performed with constant input locality. On the negative side, we show that primitives which require some form of non-malleability (such as digital signatures, message authentication, or non-malleable encryption) *cannot* be realized with constant input locality. On the positive side, assuming the intractability of certain problems from the domain of error correcting codes (namely, hardness of decoding a random linear code or the security of the McEliece cryptosystem), we obtain new constructions of one-way functions, pseudorandom generators, commitments, and semantically-secure public-key encryption schemes whose input locality is constant. Moreover, these constructions also enjoy constant *output locality*. Therefore, they give rise to cryptographic hardware that has constant-depth, constant fan-in and constant *fan-out*. As a byproduct, we obtain a pseudorandom generator whose output and input locality are both optimal (namely, 3).

7.1 Introduction

In the previous chapters we showed that, under standard cryptographic assumptions, most cryptographic primitives can be realized by functions with constant *output* locality, namely ones in which every bit of the *output* is influenced by a constant number of bits from the input. In this chapter we study the complementary question of implementing cryptographic primitives by functions in which each *input* bit affects only a constant number of output bits. This was not settled by the previous chapters. This natural question can be motivated from several distinct perspectives:

- (Theoretical examination of a common practice) A well known design principle for practical cryptosystems asserts that each input bit must affect many output bits. This principle is sometimes referred to as Confusion/Diffusion or Avalanche property. It is easy to justify this principle in the context of block-ciphers (which are theoretically modeled as pseudorandom functions or permutations), but is it also necessary in other cryptographic applications (e.g., stream ciphers)?

- (Hardware perspective) Unlike NC^0 functions, functions with both constant input locality and constant output locality can be computed by constant depth circuits with bounded fan-in and *bounded fan-out*. Hence, the parallel time complexity of such functions is constant in a wider class of implementation scenarios.

- (Complexity theoretic perspective) One can state the existence of cryptography in NC^0 in terms of average-case hardness of Constraint Satisfaction Problems in which each constraint involves a constant number of variables (k-CSPs). The new question can therefore be formulated in terms of k-CSPs with bounded occurrences of each variable. It is known that NP hardness and inapproximability results can be carried from the CSP setting to this setting [PY91, Coo71], hence it is interesting to ask whether the same phenomenon occurs with respect to cryptographic hardness as well.

Motivated by the above, we would like to understand which cryptographic tasks (if any) can be realized with constant input *and* output locality, or even with constant input locality alone. Another question considered in this chapter, is that of closing the (small) gap between positive results for cryptography with locality 4 and the impossibility of cryptography with locality 2.

7.1.1 Our Results

We provide an almost full characterization of the cryptographic tasks that can be realized by functions with constant input locality. On the negative side, we show that primitives which require some form of non-malleability (e.g., signatures, MACs, non-malleable encryption schemes) *cannot* be realized with constant (or, in some cases, even logarithmic) input locality.

On the positive side, assuming the intractability of some problems from the domain of error correcting codes, we obtain constructions of pseudorandom generators, commitments, and semantically-secure public-key encryption schemes with constant input locality and constant output locality. In particular, we obtain the following results:

- For PRGs, we answer simultaneously both of the above questions. Namely, we construct a collection[1] of PRGs whose output locality and input locality are both 3. We show that this is optimal in both output locality and input locality. Our construction is based on the intractability of decoding a random linear code. Previous constructions of PRGs, or even OWFs, (cf. [Gol00], Chapter 6) which enjoyed constant input locality and constant output locality at the same time, were based on non-standard intractability assumptions.

- We construct a non-interactive commitment scheme, in the common reference string model, in which the output locality of the commitment function is 4, and its input locality is 3. The security of this scheme also follows from the intractability of decoding a random linear code. (We can also get a non-interactive commitment scheme in the standard model under the assumption that there exists an explicit binary linear code that has a large minimal distance but is hard to decode.)

[1]All of our collections are indexed by a public random key. That is, $\{G_z\}_{z\in\{0,1\}^*}$ is a collection of PRGs if for every z the function G_z expands its input and the pair $(z, G_z(x))$ is pseudorandom for random x and z. See Definition 7.2.1.

- We construct a semantically secure public-key encryption scheme whose encryption algorithm has input locality 3. This scheme is based on the security of the McEliece cryptosystem [McE78], an assumption which is related to the intractability of decoding a random linear code, but is seemingly stronger. Our encryption function also has constant output locality, if the security of the McEliece cryptosystem holds when it is instantiated with some error correcting code whose relative distance is constant.

- We show that MACs, signatures and non-malleable symmetric or public-key encryption schemes cannot be realized by functions whose input locality is constant or, in some cases, even logarithmic in the input length. In fact, we prove that even the weakest versions of these primitives (e.g., one-time secure MACs) cannot be constructed in this model.

7.1.2 Our Techniques

Our constructions rely again on the machinery of randomized encoding. In Chapter 4 we showed that the security of most cryptographic primitives is inherited by their randomized encoding. Thus, in order to construct some cryptographic primitive $\mathcal{P}$ in some low complexity class WEAK, one can try to encode functions from a higher complexity class STRONG by functions from WEAK and then take an implementation $f \in \mathsf{STRONG}$ of the primitive $\mathcal{P}$, and replace it by its encoding $\hat{f} \in \mathsf{WEAK}$. This paradigm was used in Chapters 4, 5. (For example, we showed that STRONG can be NC^1 and WEAK can be the class of functions whose output locality is 4.)

However, it seems hard to adapt this approach to the current setting, since it is not clear whether there are non-trivial functions that can be encoded by functions with constant input locality. (In fact, we show that some very simple NC^0 functions cannot be encoded in this class.) We solve this problem by introducing a new construction of randomized encodings. Our construction shows that there exists a complexity class $\mathcal{C}$ of simple (but non-trivial) functions that can be encoded by functions with constant input locality. Roughly speaking, a function f is in $\mathcal{C}$ if each of its output bits can be written as a sum of terms over $\mathbb{F}_2$ such that each input variable of f participates in a constant number of *distinct* terms, ranging over all outputs of f. Moreover, if the algebraic degree of theses terms is constant, then f can be encoded by a function with constant input locality as well as constant output locality. (In particular, all linear functions over $\mathbb{F}_2$ admit such an encoding.)

By relying on the nice algebraic structure of intractability assumptions related to decoding random linear codes, and using techniques from Chapter 6, we construct PRGs, commitments and public-key encryption schemes in $\mathcal{C}$ whose algebraic degree is constant. Then, we use the new construction to encode these primitives, and obtain implementations whose input locality and output locality are both constant.

Interestingly, unlike previous constructions of randomized encodings, the new encoding does not have a universal simulator nor a universal decoder; that is, one should use different decoders and simulators for different functions in C. This phenomenon is inherent to the setting of constant input locality and is closely related to the fact that MACs cannot be realized in this model. See Section 7.6.2 for a discussion.

7.1.3 Previous Work

Unlike the case of NC^0 cryptography, the question of cryptography with constant input locality is relatively unexplored. (A review of the works that addressed the possibility of NC^0 cryptography appears in Section 4.1.1.) However, constructions of primitives with constant input locality are implicitly given in [Gol00, MST03]. In particular, Goldreich [Gol00] suggested an approach for constructing OWFs based on expander graphs, an approach whose conjectured security does not follow from any well-known assumption. This general construction can be instantiated by functions with constant output locality and constant input locality. In addition, Mossel et al. [MST03] constructed (non-cryptographic) ε-biased generators with (non-optimal) constant input and output locality. Also, the construction of linear-stretch PRG in NC^0 of Chapter 6, can give an NC^0 PRG with (large) constant input locality under Assumption 6.5.1, which is a non-standard assumption taken from [Ale03]. (See Item 2 in Remarks on Theorem 6.5.11.)

7.2 Preliminaries

We will consider collections of PRGs.

Definition 7.2.1 (Collection of PRGs) *Let $p(\cdot)$ be a polynomial, and let $\mathcal{G} = \{G_z\}_{z\in\{0,1\}^{p(n)}}$ be a polynomial-time computable collection of functions where $G_z : \{0,1\}^n \to \{0,1\}^{s(n)}$. Then $\mathcal{G}$ is a PRG collection, if it satisfies the following two conditions:*

1. **Expansion***: $s(n) > n$, for all $n \in \mathbb{N}$.*
2. **Pseudorandomness***: $\{z, G_z(U_n)\}_{n\in\mathbb{N}} \overset{c}{\equiv} \{z, U_{s(n)}\}_{n\in\mathbb{N}}$, where $z \leftarrow U_{p(n)}$.*

In particular, if we have a PRG of the form $G(x,z) = (G'(x,z), z)$ then we can transform it into the PRG collection $G_z(x) = G'(x,z)$. Note, that this definition falls into the category of public-coin collection (as defined in Appendix A). Moreover, in this case the public-key is simply a random string.

We will also need the definition of extractors. Recall that the min-entropy of a random variable X is defined as $\mathrm{H}_\infty(X) \overset{\text{def}}{=} \min_x \log(\frac{1}{\Pr[X=x]})$.

Definition 7.2.2 (Extractor) *A function $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^t$ is a (k,ε)-extractor if for every distribution X on $\{0,1\}^n$ with $\mathrm{H}_\infty(X) \geq k$ the distribution $\mathrm{Ext}(X, U_d)$ is ε-close to the uniform distribution over $\{0,1\}^t$.*

An important construction of extractors is based on pairwise independent hashing.

Lemma 7.2.3 (Leftover hashing lemma [HILL99]) *Let $H = \{h_z\}$ be a family of pairwise independent hash functions that map n-bit strings to t-bit strings. Then, the function $\mathrm{Ext}(x,z) = (h_z(x), z)$ is a (k,ε) extractor where $\varepsilon = 2^{-(k-t)/2}$.*

We note that pairwise independent hash functions can be defined by the mapping $h_{A,b}(x) = Ax + b$ where A is a $t \times n$ binary matrix, b is a t-bit vector and arithmetic is over $\mathbb{F}_2$.

In the following definitions, we will use k to denote a random string, which can be either a public string or a secret key. A commitment scheme in the common reference string model (CRS) is defined as follows:

Definition 7.2.4 (Commitment scheme) *A commitment scheme is a pair* (Com, Rec) *of probabilistic polynomial-time algorithms satisfying the following conditions:*

- **Viability***: For every bit* $b \in \{0,1\}$*, it holds* $\Pr_{k,r}[\textsc{Rec}_k(\textsc{Com}_k(b;r), b, r) = \textsf{reject})] \le \text{neg}(|k|)$.
- **Hiding***:* $\{(k, \textsc{Com}_k(0;r))\}_n \overset{c}{\equiv} \{(k, \textsc{Com}_k(1;r))\}_n$ *where* $k \leftarrow U_n, r \leftarrow U_{p(n)}$*, and the polynomial* $p(\cdot)$ *is the randomness complexity of* $\textsc{Com}$.
- **Binding***: For every commitment string* c*, except with negligible probability over the choice of* k*, there are no* r_0 *and* r_1 *for which* $\textsc{Rec}_k(c, 0, r_0) = \textsc{Rec}_k(c, 1, r_1) = \textsf{accept}$.

The following definition of Non-Malleable private-key encryption is based on the definition of [KY00]. Since this definition is used here for negative results, we allow ourselves to use a simplified version which is considerably weaker than the original definition.

Definition 7.2.5 (Non-malleable private-key encryption) *A* non-malleable private-key encryption scheme *is a pair* (E, D) *of probabilistic polynomial-time algorithms satisfying the following conditions:*

- **Viability***: The algorithms* E, D *satisfy*

$$\Pr[D_k(E_k(x)) \neq x)] \le \text{neg}(|k|).$$

- **Non-Malleability***: For an adversary* A*, consider the following experiment which is indexed by* n*, the size of the key: First a random* n*-bit key* k *is selected. Then,* A *outputs a message space distribution* M *which consists of strings of equal length (and represented by a probabilistic polynomial-sized circuit), and a binary relation* R *(which is also represented by a polynomial-sized circuit). Next, two random strings* $x, \tilde{x}$ *are chosen from* M*, and are encrypted under* k*. Let* $c, \tilde{c}$ *be the resulting ciphertexts. Then, the ciphertext* c *is given to* A*, which in turn outputs a ciphertext* $c' \neq c$*. The advantage of* A *is defined to be*

$$\varepsilon_A(n) = \left|\Pr[(D_k(c'), D_k(c)) \in R] - \Pr[(D_k(c'), D_k(\tilde{c})) \in R]\right|.$$

The scheme is non-malleable *if for every (non-uniform) efficient adversary* A *the advantage* $\varepsilon_A(n)$ *of* A *is negligible in* n*.*

7.3 Randomized Encoding with Constant Input Locality

In this section we will show that functions with a "simple" algebraic structure (and in particular *linear* functions over $\mathbb{F}_2$) can be encoded by functions with constant input locality. We begin with the following construction that shows how to reduce the input locality of a function which is represented as a sum of functions.

Construction 7.3.1 (Basic input locality construction) *Let*

$$f(x) \quad = \quad (a(x) + b_1(x), a(x) + b_2(x), \ldots, a(x) + b_k(x), c_1(x), \ldots, c_l(x)),$$

where $f : \mathbb{F}_2^n \to \mathbb{F}_2^{k+l}$ *and* $a, b_1, \ldots, b_k, c_1, \ldots, c_l : \mathbb{F}_2^n \to \mathbb{F}_2$*. The encoding* $\hat{f} : \mathbb{F}_2^{n+k} \to \mathbb{F}_2^{2k+l}$ *is defined by:*

$$\begin{aligned}\hat{f}(x, (r_1, \ldots, r_k)) \quad \overset{def}{=} \quad & (r_1 + b_1(x), r_2 + b_2(x), \ldots, r_k + b_k(x), \\ & a(x) - r_1, r_1 - r_2, \ldots, r_{k-1} - r_k, \\ & c_1(x), \ldots, c_l(x)).\end{aligned}$$

Note that after the transformation the function $a(x)$ appears only once and therefore the locality of the input variables that appear in a is reduced. In addition, the locality of all the other original input variables does not increase.[2] For example, applying the locality construction to the function $f(x) = (x_1x_2 + x_2, x_1x_2 + x_2x_3, x_1x_2 + x_3, x_3)$ results in the encoding $\hat{f}(x,r) = (r_1 + x_2, r_2 + x_2x_3, r_3 + x_3, x_1x_2 - r_1, r_1 - r_2, r_2 - r_3, x_3)$. Hence, in this case it reduces the locality of x_1 from 3 to 1.

Lemma 7.3.2 (Input locality lemma) *Let f and $\hat{f}$ be as in Construction 7.3.1. Then, $\hat{f}$ is a perfect randomized encoding of f.*

Proof: The encoding $\hat{f}$ is stretch-preserving since the number of random inputs equals the number of additional outputs (i.e., k). Moreover, given a string $\hat{y} = \hat{f}(x,r)$ we can decode the value of $f(x)$ as follows: To recover $a(x) + b_i(x)$, compute the sum $y_i + y_{k+1} + y_{k+2} + \ldots + y_{k+i}$; To compute $c_i(x)$, simply take y_{2k+i}. This decoder never errs.

Fix some $x \in \{0,1\}^n$. Let $y = f(x)$ and let $\hat{y}$ denote the distribution $\hat{f}(x, U_k)$. To prove perfect privacy, note that: (1) the last l bits of $\hat{y}$ are fixed and equal to $y_{[k+1\ldots k+l]}$; (2) the first k bits of $\hat{y}$ are independently uniformly distributed; (3) the remaining bits of $\hat{y}$ are uniquely determined by y and $\hat{y}_1, \ldots, \hat{y}_k$. To see (3), observe that, by the definition of $\hat{f}$, we have $\hat{y}_{k+1} = y_1 - \hat{y}_1$; and for every $1 < i \le k$, we also have $\hat{y}_{k+i} = y_i - \hat{y}_i - \sum_{j=1}^{i-1} \hat{y}_{k+j}$.

Hence, define a perfect simulator as follows. Given $y \in \{0,1\}^{k+l}$, the simulator S chooses a random string r of length k, and outputs $(r, s, y_{[k+1\ldots k+l]})$, where $s_1 = y_1 - r_1$ and $s_i = y_i - r_i - \sum_{j=0}^{i-1} s_j$ for $1 < i \le k$. This simulator is also balanced as each of its outputs is a linear function that contains a fresh random bit. (Namely, the output bit $S(y;r)_i$ depends on: (1) r_i if $1 \le i \le k$; or (2) y_{i-k} if $k+1 \le i \le 2k+l$.) ■

An additive representation of a function $f : \mathbb{F}_2^n \to \mathbb{F}_2^l$ is a representation in which each output bit is written as as a sum (over $\mathbb{F}_2$) of functions of the input x. That is, each output bit f_i can be written as $f_i(x) = \sum_{a \in T_i} a(x)$, where T_i is a set of boolean functions over n variables. We specify such an additive representation by an l-tuple $(T_1, \ldots, T_l)$ where T_i is a set of boolean functions $a : \mathbb{F}_2^n \to \mathbb{F}_2$. We assume, without loss of generality, that none of the T_i's contains the constant functions 0 or 1. The following measures are defined with respect to a given additive representation of f. For a function $a : \mathbb{F}_2^n \to \mathbb{F}_2$, define the *multiplicity* of a to be the number of T_i's in which a appears, i.e., $\#a = |\{T_i \mid a \in T_i\}|$. For a variable x_j, we define the *rank* of x_j to be the number of different boolean functions a which depend on x_j and appear in some T_i. That is, $\mathrm{rank}(x_j) = |\{a : \mathbb{F}_2^n \to \mathbb{F}_2 \mid a \text{ depends on } x_j, a \in T_1 \bigcup \ldots \bigcup T_l\}|$.

Theorem 7.3.3 *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2^l$ be a function, and fix some additive representation $(T_1, \ldots, T_l)$ for f. Then f can be perfectly encoded by a function $\hat{f} : \mathbb{F}_2^n \times \mathbb{F}_2^m \to \mathbb{F}_2^s$ such that the following hold:*

1. *The input locality of every x_j in $\hat{f}$ is at most $\mathrm{rank}(x_j)$, and the input locality of the random inputs r_i of $\hat{f}$ is at most 3.*

2. *If the output locality of f is i, then the output locality of $\hat{f}$ is $\max(i, 2)$.*

3. *The randomness complexity of $\hat{f}$ is $m = \sum_{a \in T} \#a$, where $T = \bigcup_{i=1}^{l} T_i$.*

[2]Note that it is trivial to obtain a deterministic encoding that satisfies only the first property by letting $\hat{f}(x) = (a(x) + b_1(x), b_2(x) - b_1(x), \ldots, b_k(x) - b_1(x), c_1(x), \ldots, c_l(x))$.

Proof: We will use the following convention. The additive representation of a function $\hat{g}$ resulting from applying Construction 7.3.1 to a function g is the (natural) representation induced by the original additive representation of g. We construct $\hat{f}$ iteratively via the following process.

- Let $f^{(0)} = f, i = 0$.
- For $j = 1, \ldots, n$ do the following:
  - while there exists a function a in $f^{(i)}$ that depends on x_j, whose multiplicity is greater than 1, apply Construction 7.3.1 to $f^{(i)}$. Let $f^{(i+1)}$ be the resulting encoding and let $i = i + 1$.
- Let $\hat{f} = f^{(i)}$.

By Lemma 7.3.2, the function $f^{(i)}$ perfectly encodes the function $f^{(i-1)}$, hence by the composition property of randomized encodings (Lemma 3.2.3), the final function $\hat{f}$ perfectly encodes f. The first item of the theorem follows from the following observations: (1) In each iteration the input locality and the rank of each original variable x_j do not increase. (2) The multiplicity in $\hat{f}$ of every function a that depends on some original input variable x_j is 1. (3) The input locality of the random inputs which are introduced by the locality construction is at most 3. The last two items of the theorem follow directly from the definition of Construction 7.3.1 and the construction of $\hat{f}$.

■

Remarks on Theorem 7.3.3.

1. By Theorem 7.3.3, every linear function admits an encoding of constant input locality, since each output bit can be written as a sum of degree 1 monomials. More generally, every function f whose canonic representation as a sum of monomials (i.e., each output bit is written as a sum of monomials) includes a constant number of monomials per input bit can be encoded by a function of constant input locality.

2. Interestingly, Construction 7.3.1 does not provide a universal encoding for any natural class of functions (e.g., the class of linear functions mapping n bits into l bits). This is contrasted with previous constructions of randomized encoding with constant output locality (cf. Section 4.2). In fact, in Section 7.6.1 we prove that there is no universal encoding with constant input locality for the class of linear function $L : \mathbb{F}_2^n \to \mathbb{F}_2$.

3. When Theorem 7.3.3 is applied to a function family $f : \{0,1\}^n \to \{0,1\}^{l(n)}$ then the resulting encoding is uniform whenever the additive representation $(T_1, \ldots, T_l)$ is polynomial-time computable.

4. In Section 7.6.1, we show that Theorem 7.3.3 is tight in the sense that for each integer i we can construct a function f in which the rank of x_1 is i, and in every encoding $\hat{f}$ of f the input locality of x_1 is at least i.

In some cases we can combine Theorem 7.3.3 and the output-locality construction (Construction 4.2.4) to derive an encoding which enjoys low input locality and output locality at the same time. In particular, we will use the following lemma which is a refinement of Lemma 4.2.5.

Lemma 7.3.4 (implicit in Chapter 4) *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2^l$ be a function such that each of its output bits can be written as sum of monomials of degree d. Then, we can perfectly encode f by a function $\hat{f}$ such that:*

- $\hat{f} \in \mathrm{Local}^{d+1}$.
- *The rank of every original variable x_i in $\hat{f}$ is equal to the rank of x_i in f.*
- *The new variables introduced by $\hat{f}$ appear only in monomials of degree 1; hence their rank is 1.*

By combining Lemma 7.3.4 with Theorem 7.3.3 we get:

Corollary 7.3.5 *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2^l$ be a function. Fix some additive representation for f in which each output bit is written as a sum of monomials of degree (at most) d and the* rank *of each variable is at most ρ. Then, f can be perfectly encoded by a function $\hat{f}$ of input locality $\max(\rho, 3)$ and output locality $d+1$. Moreover, the resulting encoding is uniform whenever the additive representation is polynomial-time computable.*

Proof: First, by Lemma 7.3.4, we can perfectly encode f by a function $f' \in \mathrm{Local}^{d+1}$ without increasing the rank of the input variables of f. Next, we apply Theorem 7.3.3 and perfectly encode f' by a function $\hat{f} \in \mathrm{Local}^{d+1}_{\max(\rho,3)}$. By the composition property of randomized encodings (Lemma 3.2.3), the resulting function $\hat{f}$ perfectly encodes f. Finally, the proofs of Theorem 7.3.3 and Lemma 7.3.4 both allow to efficiently transform an additive representation of the function f into an encoding $\hat{f}$ in $\mathrm{Local}^{d+1}_{\max(\rho,3)}$. Hence, the uniformity of f is inherited by $\hat{f}$. ■

We remark that Theorem 7.3.3 as well as Lemma 7.3.4 generalize to any finite field $\mathbb{F}$. Hence, so does Corollary 7.3.5.

7.4 Primitives with Constant Input Locality and Output Locality

7.4.1 Main Assumption: Intractability of Decoding Random Linear Code

Our positive results are based on the intractability of decoding a random linear code. In the following we introduce and formalize this assumption.

An (m, n, δ) *binary linear code* is a n-dimensional linear subspace of $\mathbb{F}_2^m$ in which the Hamming distance between each two distinct vectors (codewords) is at least δm. We refer to the ratio n/m as the *rate* of the code and to δ as its (relative) *distance*. Such a code can be defined by an $m \times n$ *generator matrix* whose columns span the space of codewords. It follows from the Gilbert–Varshamov bound that whenever $n/m < 1 - \mathrm{H}_2(\delta) - \varepsilon$, almost all $m \times n$ generator matrices form (m, n, δ)-linear codes. Formally,

Fact 7.4.1 ([Var57]) *Let $0 < \delta < 1/2$ and $\varepsilon > 0$. Let $n/m \leq 1 - \mathrm{H}_2(\delta) - \varepsilon$. Then, a randomly chosen $m \times n$ generator matrix generates an (m, n, δ) code with probability $1 - 2^{-(\varepsilon/2)m}$.*

A proof of the above version of the Gilbert–Varshamov bound can be found in [Sud02, Lecture 5]. For code length parameter $m = m(n)$, and noise parameter $\mu = \mu(n)$, we will consider the following "decoding game". Pick a random $m \times n$ matrix C representing a linear code, and a random information word x. Encode x with C and transmit the resulting codeword $y = Cx$ over a binary symmetric channel in which every bit is flipped with probability μ. Output the noisy codeword $\tilde{y}$ along with the code's description C. The adversary's task is to find the information word x. We say that the above game is intractable if every polynomial-time adversary wins in the above game with no more than negligible probability in n.

Definition 7.4.2 *Let $m(n) \leq \mathrm{poly}(n)$ be a code length parameter, and $0 < \mu(n) < 1/2$ be a noise parameter. The problem $\mathrm{CODE}(m, \mu)$ is defined as follows:*

- **Input:** *$(C, Cx + e)$, where C is an $m(n) \times n$ random binary generator matrix, $x \leftarrow U_n$, and $e \in \{0,1\}^m$ is a random error vector in which each entry is chosen to be 1 with probability μ (independently of other entries), and arithmetic is over $\mathbb{F}_2$.*
- **Output:** *x.*

We say that $\mathrm{CODE}(m, \mu)$ is intractable *if every (non-uniform) polynomial-time adversary A solves the problem with probability negligible in n.*

We note that $\mathrm{CODE}(m, \mu)$ becomes harder when m is decreased and μ is increased, as we can always add noise or ignore the suffix of the noisy codeword. Formally,

Proposition 7.4.3 *Let $m'(n) \leq m(n)$ and $0 < \mu(n) \leq \mu'(n) < 1/2$ for every n. Then, if $\mathrm{CODE}(m, \mu)$ is intractable, so is $\mathrm{CODE}(m', \mu')$.*

Proof: Fix n and let $m' = m'(n), m = m(n), \mu = \mu(n)$ and $\mu' = \mu'(n)$. We reduce the problem $\mathrm{CODE}(m, \mu)$ to $\mathrm{CODE}(m', \mu')$ as follows. Given an input (C, y) for $\mathrm{CODE}(m, \mu)$ (i.e., C is an $m \times n$ binary matrix and y is an m-bit vector), we construct the pair (C', y') by letting C' denote the $m' \times n$ binary matrix that contains the first m' rows of C, and $y' \in \{0,1\}^{m'}$ be the vector resulting by taking the first m' entries of y and adding (over $\mathbb{F}_2$) a random vector $r \in \{0,1\}^{m'}$ in which each entry is chosen to be 1 (independently of other entries) with probability $(\mu' - \mu)/(1 - 2\mu)$.

Suppose that (C, y) is drawn from the input distribution of $\mathrm{CODE}(m, \mu)$, that is, C is random matrix and $y = Cx + e$ where $x \leftarrow U_n$, and $e \in \{0,1\}^m$ is a random error vector of noise rate μ. Then, the pair (C', y') can be written as $(C', C'x + e')$ where $e' = e + r$ is a random noise vector of rate

$$\mu \cdot (1 - \frac{\mu' - \mu}{1 - 2\mu}) + (1 - \mu)\frac{\mu' - \mu}{1 - 2\mu} = \mu + \frac{(1 - 2\mu)(\mu' - \mu)}{1 - 2\mu} = \mu'.$$

Hence, given an algorithm A that solves $\mathrm{CODE}(m', \mu')$, we can find the information word x by running A on (C', y'). ■

Typically, we let $m(n) = O(n)$ and μ be a constant such that $n/m(n) < 1 - \mathrm{H}_2(\mu + \varepsilon)$ where $\varepsilon > 0$ is a constant. Hence, by Fact 7.4.1, the random code C is, with overwhelming probability, an $(m, n, \mu + \varepsilon)$ code. Note that, except with negligible probability, the noise vector flips less than $\mu + \varepsilon$ of the bits of y. In this case, the fact that the noise is random (rather than adversarial) guarantees, by Shannon's coding theorem (for random linear codes), that x will be unique with overwhelming probability. That is, roughly speaking, we assume that it is intractable to correct μn *random* errors in a random linear code of relative distance $\mu + \varepsilon > \mu$. The plausibility of such an assumption is supported by the fact that a successful adversary would imply a major breakthrough in coding theory. Similar assumptions were put forward in [GKL93, BFKL94, Gol01a]. We mention that the best known algorithm for $\mathrm{CODE}(m, \mu)$, due to [BKW03, Lyu05], runs in time $2^{O(n/\log\log n)}$ and requires m to be super-linear, i.e., $m = n^{1+\alpha}$. When $m = O(n)$ (and μ is constant), the problem is only known to be solved in *exponential* time.

We now show that distinguishing the distribution $(C, Cx + e)$ from the uniform distribution reduces to decoding x. A similar lemma was proved by Blum et al. [BFKL94, Theorem 13]. However, their version does not preserve the length of the codewords. Namely, they show that the hardness of decoding random linear code with codewords of length $m(n)$ implies the pseudorandomness of the distribution $(C, Cx + e)$ in which the length of the codewords is *polynomially smaller* than $m(n)$.

Lemma 7.4.4 *Let $m(n)$ be a code length parameter, and $\mu(n)$ be a noise parameter. If $\mathrm{CODE}(m, \mu)$ is intractable then the distribution $(C, Cx+e)$ is pseudorandom, where $C \leftarrow U_{m(n)\times n}$, $x \leftarrow U_n$, and $e \in \{0,1\}^{m(n)}$ is a random error vector of noise rate μ.*

Proof: Assume that $\mathrm{CODE}(m, \mu)$ is intractable. Then, by the Goldreich-Levin hardcore bit theorem [GL89], given $(C, Cx + e)$ and a random n-bit vector r, an efficient adversary cannot compute $\langle r, x\rangle$ with probability greater than $\frac{1}{2} + \mathrm{neg}(n)$. Assume, towards a contradiction, that there exists an efficient distinguisher $A = \{A_n\}$ and a polynomial $p(\cdot)$ such that

$$\Pr[A_n(C, Cx + e) = 0] - \Pr[A_n(U_{m(n)\times n}, U_m) = 0] > 1/p(n),$$

for infinitely many n's. We will use A_n to construct an efficient adversary A'_n that breaks the security of the Goldreich-Levin hardcore bit. Given $(C, y = Cx + e)$ and a random n-bit vector r, the adversary A'_n chooses a random m-bit vector s and computes a new $m(n) \times n$ binary matrix $C' \overset{\text{def}}{=} C - s\cdot r^T$, where r^T denotes the transpose of r. Now A'_n applies the distinguisher A_n to (C', y) and outputs his answer. Before we analyze the success probability of A'_n we need two observations: (1) the matrix C' is a random $m(n) \times n$ binary matrix; and (2) $y = Cx + e = C'x + s \cdot r^T \cdot x + e = C'x + s \cdot \langle r, x\rangle + e$. Hence, when $\langle r, x\rangle = 0$ it holds that $(C', y) = (C', C'x + e)$, and when $\langle r, x\rangle = 1$ we have $(C', y) = (C', C'x + e + s) \equiv (C', U_m)$, where U_m is independent of C'. Therefore we have

$$\begin{aligned}
\Pr[A'_n(C, Cx + e, r) = \langle x, r\rangle] &= \Pr[A'_n(C, Cx + e, r) = 0 | \langle x, r\rangle = 0] \cdot \Pr[\langle x, r\rangle = 0] \\
&\quad + \Pr[A'_n(C, Cx + e, r) = 1 | \langle x, r\rangle = 1] \cdot \Pr[\langle x, r\rangle = 1] \\
&= \frac{1}{2} \cdot (\Pr[A_n(C', C'x + e) = 0] + 1 - \Pr[A_n(C', U_m) = 0]) \\
&\geq \frac{1}{2} + \frac{1}{2p(n)},
\end{aligned}$$

where the last inequality holds for infinitely many n's. Thus, we derive a contradiction to the security of the GL-hardcore bit. ■

7.4.2 Pseudorandom Generator in Local_3^3

A pseudorandom generator (PRG) is an efficiently computable function G which expands its input and its output distribution $G(U_n)$ is pseudorandom. An efficiently computable collection of functions $\{G_z\}_{z\in\{0,1\}^*}$ is a PRG collection if for every z, the function G_z expands its input and the pair $(z, G_z(x))$ is pseudorandom for random x and z. (See Section 7.2 for formal definitions.) We show that pseudorandom generators (and therefore also one-way functions and one-time symmetric encryption schemes) can be realized by $\mathrm{Local}^{O(1)}_{O(1)}$ functions. Specifically, we get a PRG in Local^{3}_{3}. Recall that, by the tractability of 2-SAT, it is impossible to construct a PRG (and even OWF) in Local^2 [CM01, Gol00]. In Section 7.5.4 we also prove that there is no PRG in Local_2. Hence, our PRG has optimal input locality as well as optimal output locality.

We rely on the following assumption.

Intractability Assumption 7.4.5 *The problem* $\mathrm{CODE}(6n, 1/4)$ *is intractable.*

Note that the code considered here is of rate $n/m = 1/6$ which is strictly smaller than $1 - H_2(1/4)$. Therefore, except with negligible probability, its relative distance is larger than $1/4$. Hence, the above assumption roughly says that it is intractable to correct $n/4$ random errors in a random linear code of relative distance $1/4 + \varepsilon$, for some constant $\varepsilon > 0$.

Let $m(n) = 6n$. Let $C \leftarrow U_{m(n)\times n}$, $x \leftarrow U_n$ and $e \in \{0,1\}^m$ be a random error vector of rate $1/4$, that is, each of the entries of e is 1 with probability $1/4$ (independently of the other entries). By Lemma 7.4.4, the distribution $(C, Cx + e)$ is pseudorandom under the above assumption. Since the noise rate is $1/4$, it is natural to sample the noise distribution e by using $2m$ random bits $r_1, \ldots, r_{2m}$ and letting the i-th bit of e be the product of two fresh random bits, i.e., $e_i = r_{2i-1} \cdot r_{2i}$. We can now define the mapping $f(C, x, r) = (C, Cx + e(r))$ where $e(r) = (r_{2i-1} \cdot r_{2i})_{i=1}^{m}$. The output distribution of f is pseudorandom, however, f is not a PRG since it does not expand its input. In Chapter 6, we showed how to bypass this problem by applying a randomness extractor (see Definition 7.2.2). Namely, the following function was shown to be a PRG: $G(C, x, r, s) = (C, Cx + e(r), \mathrm{Ext}(r, s))$. Although the setting of parameters in Chapter 6 is different than ours, a similar solution works here as well. We rely on the leftover hashing lemma (Lemma 7.2.3) and base our extractor on a family of pairwise independent hash functions (which is realized by the mapping $x \mapsto Ax + b$ where A is a random matrix and b is a random vector).[3]

Construction 7.4.6 *Let* $m = 6n$ *and let* $t = \lceil 7.1 \cdot n \rceil$. *Define the function*

$$G(x, C, r, A, b) \overset{def}{=} (C, Cx + e(r), Ar + b, A, b),$$

where $x \in \{0,1\}^n$, $C \in \{0,1\}^{m\times n}$, $r \in \{0,1\}^{2m}$, $A \in \{0,1\}^{t\times 2m}$, *and* $b \in \{0,1\}^t$.

Theorem 7.4.7 *Under Assumption 7.4.5, the function G defined in Construction 7.4.6 is a PRG.*

Before proving Theorem 7.4.7, we need the following claim (which is similar to Lemma 6.5.6).

Claim 7.4.8 *Let* $[r|e(r)]$ *denote the distribution of* r *given the outcome of* $e(r)$. *Then,*

$$\Pr_{r\leftarrow U_{2m}}[\mathrm{H}_\infty([r|e(r)]) \geq 1.17m] \geq 1 - \exp(-\Omega(m)).$$

[3] We remark that in Chapter 6 one had to rely on a specially made extractor in order to maintain the large stretch of the PRG. In particular, the leftover hashing lemma could not be used there.

Proof: We view $e(r)$ as a sequence of m independent Bernoulli trials, each with a probability 0.25 of success. Recall that r is composed of m pairs of bits, and that the i-th bit of $e(r)$ is 1 if and only if r_{2i-1} and r_{2i} are both equal to 1. Hence, whenever $e(r)_i = 0$, the pair (r_{2i-1}, r_{2i}) is uniformly distributed over the set $\{00, 01, 10\}$. Consider the case in which at most $0.26m$ components of $e(r)$ are ones. By a Chernoff bound, the probability of this event is at least $1 - \exp(-\Omega(m))$. In this case, r is uniformly distributed over a set of size at least $3^{0.74m}$. Hence, conditioning on the event that at most $0.26m$ components of $e(r)$ are ones, the min-entropy of $[r|e(r)]$ is at least $0.74m \log(3) > 1.17m$. ■

We can now prove Theorem 7.4.7.

Proof of Theorem 7.4.7: Let $m = 6n$ and $t = 7.01n$. First we show that G expands its input. Indeed, the difference between the output length and the input length is: $m + t - (n + 2m) = 0.01n > 0$.

Let $x \leftarrow U_n$, $C \leftarrow U_{m \cdot n}$, $r \leftarrow U_{2m}$, $A \leftarrow U_{t \cdot 2m}$ and $b \leftarrow U_t$. We prove that the distribution $G(x, C, r, A, b)$ is pseudorandom. By Lemma 7.2.3, Fact 2.2.4 and Claim 7.4.8, we have that

$$\begin{aligned} \mathrm{SD}((e(r), Ar + b, A, b), (e(r), U_{t+2tm+m})) &\leq 2^{-(1.17m-t)/2} + \exp(-\Omega(m)) \\ &= 2^{-0.005n} + \exp(-\Omega(n)) \leq \exp(-\Omega(n)). \end{aligned}$$

Hence, by Fact 2.2.3 and Assumption 7.4.5, we have

$$(C, Cx + e(r), Ar + b, A, b) \overset{\mathrm{s}}{\equiv} (C, Cx + e(r), U_{t+2tm+m}) \overset{\mathrm{c}}{\equiv} U_{mn+m+t+2tm+m},$$

which completes the proof. ■

From now on, we fix the parameters m, t according to Construction 7.4.6. We can redefine the above construction as a collection of PRGs by letting C, A, b be the keys of the collection. Namely,

$$G_{C,A,b}(x, r) = (Cx + e(r), Ar + b).$$

We can now prove the main theorem of this section.

Theorem 7.4.9 *Under Assumption 7.4.5, there exists a collection of pseudorandom generators* $\{G_z\}_{z \in \{0,1\}^{p(n)}}$ *in* Local_3^3. *Namely, for every* $z \in \{0,1\}^{p(n)}$, *it holds that* $G_z \in \mathrm{Local}_3^3$.

Proof: Fix C, A, b and write each output bit of $G_{C,A,b}(x, r)$ as a sum of monomials. Note that in this case, each variable x_i appears only in degree 1 monomials, and each variable r_i appears only in the monomial $r_{2i-1}r_{2i}$ and also in degree 1 monomials. Hence, the rank of each variable is at most 2. Moreover, the (algebraic) degree of each output bit of $G_{C,A,b}$ is at most 2. Therefore, by Corollary 7.3.5, we can perfectly encode the function $G_{C,A,b}$ by a function $\hat{G}_{C,A,b}$ in Local_3^3. In Chapter 4 we showed that a uniform perfect encoding of a PRG is also a PRG (see Lemma 4.5.4). Thus, we get a collection of PRGs in Local_3^3. ■

Remark 7.4.10 (Symmetric Encryption in Local_3^3**)** We can rely on Theorem 7.4.9 to obtain a one-time semantically-secure symmetric encryption scheme (E, D) whose encryption algorithm is in Local_3^3. First, we instantiate Construction 5.2.4 with the PRG of Theorem 7.4.9. This gives an encryption algorithm E_k with degree 2 and rank 2. Now we can apply Corollary 7.3.5 and perfectly encode the function E_k by a function $\hat{E}_k$ in Local_3^3. Since randomized encoding preserves semabtic security (Lemma 4.7.2), the resulting function defines an encryption algorithm of a one-time semantically-secure symmetric encryption scheme. (This scheme allows to encrypt an arbitrary polynomially long message with a short key.) A similar approach can be also used to give multiple message security, at the price of requiring the encryption and decryption algorithms to maintain a synchronized *state*. The results of Section 7.4.4 give a direct construction of public-key encryption (hence also symmetric encryption) with constant input locality under the stronger assumption that the McEliece cryptosystem is one-way secure.

7.4.3 Commitment in Local_3^4

We will consider a non-interactive commitment scheme in the common reference string (CRS) model. In such a scheme, the sender and the receiver share a common public random key k (that can be selected once and be used in many invocations of the scheme). To commit to a bit b, the sender computes the commitment function $\textsc{Com}_k(b, r)$ that outputs a commitment c using the randomness r, and sends the output to the receiver. To open the commitment, the sender sends the randomness r and the committed bit b to the receiver who checks whether the opening is valid by computing the function $\textsc{Rec}_k(c, b, r)$. The scheme should be both (computationally) hiding and (statistically) binding. Hiding requires that $c = \textsc{Com}_k(b, r)$ keep b computationally secret. Binding means that, except with negligible probability over the choice of the random public key, it is impossible for the sender to open its commitment in two different ways. A formal definition is given in Section 7.2.

We construct a commitment scheme in Local_3^4, i.e., a commitment of input locality 3 and output locality 4. Let c be a constant that satisfies $c > \frac{1}{1-H_2(1/4)}$. Let $m = m(n) = \lceil cn \rceil$. Then, by Fact 7.4.1, a random $m \times n$ generator matrix generates, except with negligible probability (i.e., $2^{-\Omega(m)} = 2^{-\Omega(n)}$), a code whose relative distance is $1/4 + \varepsilon$, for some constant $\varepsilon > 0$. The public key of our scheme will be a random $m(n) \times n$ generator matrix C. To commit to a bit b, we first choose a random information word $x \in \{0,1\}^n$ and hide it by computing $Cx + e$, where $e \in \{0,1\}^m$ is a noise vector of rate 1/8, and then take the exclusive-or of b with a hardcore bit $\beta(x)$ of the above function. That is, we send the receiver the value $(Cx + e, b + \beta(x))$. In particular, we can use the Goldreich-Levin [GL89] hardcore bit and get

$$\textsc{Com}_C(b, (x, r, s)) = (Cx + e(r), s, b + \langle x, s \rangle),$$

where r is a random $3m$-bit string, $e(r) = (r_1r_2r_3, r_4r_5r_6, \ldots, r_{3m-2}r_{3m-1}r_{3m})$, s is a random n-bit string and $\langle \cdot, \cdot \rangle$ denotes inner product (over $\mathbb{F}_2$). Assuming that $\mathrm{CODE}(m, 1/8)$ is intractable, this commitment hides the committed bit b. (This is so because $\langle x, s \rangle$ is unpredictable given $(C, Cx + e, s)$, cf. [Gol01a, Construction 4.4.2].) Suppose that the relative distance of C is indeed $1/4 + \varepsilon$. Then, if e contains no more than $1/8 + \varepsilon/2$ ones, x is uniquely determined by $Cx + e$. Of course, the sender might try to cheat and open the commitment ambiguously by claiming that the weight of the error vector is larger than $1/8 + \varepsilon/2$. Hence, we let the receiver verify that the Hamming weight of the noise vector e given to him by the sender in the opening phase is indeed smaller than $1/8 + \varepsilon/2$. This way, the receiver will always catch a cheating sender (assuming that C is indeed a good code). Note that an honest sender will be rejected only if its randomly chosen noise vector is heavier than $1/8 + \varepsilon/2$, which, by a Chernoff bound, happens with negligible probability (i.e., $\exp(-\Omega(m)) = \exp(-\Omega(n))$) as the noise rate is 1/8. Hence, the pair $(\textsc{Com}, \textsc{Rec})$ defined above is indeed a commitment scheme. When C is fixed, the rank and algebraic degree of the function $\textsc{Com}_C$ are 2 and 3 (with respect to the natural representation as a sum of monomials). Hence, by Corollary 7.3.5, we can encode $\textsc{Com}_C$ by a function $\hat{\textsc{Com}}_C \in \mathrm{Local}_3^4$. By the results of Section 4.8.2, this encoding is also a commitment scheme. Summarizing, we have:

Theorem 7.4.11 *Let c be a constant that satisfies $c > \frac{1}{1-H_2(1/4)}$, and $m = m(n) = \lceil cn \rceil$. If $\mathrm{CODE}(m, 1/8)$ is intractable, then there exists a commitment scheme $(\textsc{Com}, \textsc{Rec})$ in Local_3^4; i.e., for every public key C, we have $\textsc{Com}_C \in \mathrm{Local}_3^4$.*

We remark that we can eliminate the use of the CRS by letting C be a generator matrix of some fixed error correcting error whose relative distance is large (i.e., 1/4 or any other constant) in which decoding is intractable. For example, one might use the dual of a BCH code.

7.4.4 Semantically Secure Public-Key Encryption in $\mathrm{Local}_3^{O(1)}$

We construct a semantically-secure public-key encryption scheme (PKE) whose encryption algorithm is in $\mathrm{Local}_{O(1)}^{O(1)}$ (see Definition 4.7.1). Our scheme is based on the McEliece cryptosystem [McE78]. We begin by reviewing the general scheme proposed by McEliece.

- **System parameters:** Let $m(n) : \mathbb{N} \to \mathbb{N}$, where $m(n) > n$, and $\mu(n) : \mathbb{N} \to (0, 1)$. For every $n \in \mathbb{N}$, let $\mathcal{C}_n$ be a set of generating matrices of $(m(n), n, 2(\mu(n) + \varepsilon))$ codes that have a (universal) efficient decoding algorithm D that, given a generating matrix from $\mathcal{C}_n$, can correct up to $(\mu(n) + \varepsilon) \cdot m(n)$ errors, where $\varepsilon > 0$ is some constant.[4] We also assume that there exists an efficient sampling algorithm that samples a generator matrix of a random code from $\mathcal{C}_n$.

- **Key Generation:** Given a security parameter 1^n, use the sampling algorithm to choose a random code from $\mathcal{C}_n$ and let C be its generating matrix. Let $m = m(n)$ and $\mu = \mu(n)$. Choose a random $n \times n$ non-singular matrix S over $\mathbb{F}_2$, and a random $m \times m$ permutation matrix P. Let $C' = P \cdot C \cdot S$ be the public key and P, S, D_C be the private key where D_C is the efficient decoding algorithm of C.

- **Encryption:** To encrypt $x \in \{0, 1\}^n$ compute $c = C'x + e$ where $e \in \{0, 1\}^m$ is an error vector of noise rate μ.

- **Decryption:** To decrypt a ciphertext c, compute $P^{-1}y = P^{-1}(C'x + e) = CSx + P^{-1}e = CSx + e'$ where e' is a vector whose weight equals to the weight of e (since P^{-1} is also a permutation matrix). Now, use the decoding algorithm D to recover the information word Sx (i.e., $D(C, CSx + P^{-1}e) = Sx$). Finally, to get x multiply Sx on the left by S^{-1}.

By Chernoff bound, the weight of the error vector e is, except with negligible probability, smaller than $(\mu+\varepsilon) \cdot m$ and so the decryption algorithm almost never errs. As for the security of the scheme, it is not hard to see that the scheme is *not* semantically secure. (For example, it is easy to verify that a ciphertext c is an encryption of a given plaintext x by checking whether the weight of $c - Cx$ is approximately μn.)

[4]In fact, we may allow ε to decrease with n. In such case, we might get a non-negligible decryption error. This can be fixed (without increasing the rank or the degree of the encryption function) by repeating the encryption with independent fresh randomness. Details omitted.

However, the scheme is conjectured to be a one-way cryptosystem; namely, it is widely believed that, for proper choice of parameters, any efficient adversary fails with probability $1 - \text{neg}(n)$ to recover x from $(c = C'x + e, C')$ where x is a random n-bit string. (In other words, the McEliece cryptosystem is considered to be a collection of trapdoor one-way functions which is almost 1-1 with respect to its first argument; i.e., x.)

Suppose that the scheme is indeed one-way with respect to the parameters $m(n), \mu(n)$ and $\mathcal{C}_n$. Then, we can convert it into a semantically secure public-key encryption scheme by extracting a hardcore predicate and xoring it with a 1-bit plaintext b (this transformation is similar to the one used for commitments in the previous section). That is, we encrypt the bit b by the ciphertext $(C'x+e, s, \langle s, x\rangle + b)$ where x, s are random n-bit strings, and e is a noise vector of rate μ. (Again, we use the Goldreich-Levin hardcore predicate [GL89].) To decrypt the message, we first compute x, by invoking the McEliece decryption algorithm, and then compute $\langle s, x\rangle$ and xor it with the last entry of the ciphertext. We refer to this scheme as the *modified* McEliece public-key encryption scheme. If the McEliece cryptosystem is indeed one-way, then $\langle s, x\rangle$ is pseudorandom given $(C', C'x + e, s)$, and thus the modified McEliece public-key is semantically secure. Formally,

Lemma 7.4.12 *If the McEliece cryptosystem is one-way with respect to the parameters $m(n), \mu(n)$ and $\mathcal{C}_n$, then the* modified *McEliece PKE is semantically secure with respect to the same parameters.*

The proof of this lemma is essentially the same as the proof of [Gol04, Prop. 5.3.14].

Let $\mu(n) = 2^{-t(n)}$. Then, we can sample the noise vector e by using the function $e(r) = \left(\prod_{j=1}^{t} r_{t\cdot(i-1)+j}\right)_{i=1}^{m(n)}$ where r is a $t(n) \cdot m(n)$ bit string. In this case, we can write the encryption function of the modified McEliece as $E_{C'}(b, x, r, s) = (C'x + e(r), s, \langle x, s\rangle + b)$.

The rank of each variable of this function is at most 2, and its algebraic degree is at most $t(n)$. Hence, by Corollary 7.3.5, we can encode it by a function $\hat{E} \in \text{Local}_3^{t(n)+1}$, i.e., the output locality of $\hat{E}$ is $t(n) + 1$ and its input locality is 3. In Lemma 4.7.2 we showed that randomized encoding preserves the security of PKE. Namely, if (G, E, D) is a semantically secure PKE then $(G, \hat{E}, \hat{D})$ is also an encryption scheme where $\hat{E}$ is an encoding of E, $\hat{D}(c) = D(B(c))$ and B is the decoder of the encoding. Hence we have,

Theorem 7.4.13 *If the McEliece cryptosystem is one-way with respect to the parameters $m(n)$, $\mu(n) = 2^{-t(n)}$ and $\mathcal{C}_n$, then there exists a semantically secure PKE whose encryption algorithm is in* $\text{Local}_3^{t(n)}$.

The scheme we construct encrypts a single bit, however we can use concatenation to derive a PKE for messages of arbitrary (polynomial) length without increasing the input and output locality. Theorem 7.4.13 gives a PKE with constant output locality whenever the noise rate μ is constant. Unfortunately, the binary classical Goppa Codes, which are commonly used with the McEliece scheme [McE78], are known to have an efficient decoding only for *subconstant* noise rate. Hence, we cannot use them for the purpose of achieving constant output locality and constant input locality simultaneously. Instead, we suggest using algebraic-geometric (AG) codes which generalize the classical Goppa Codes and enjoy an efficient decoding algorithm for constant noise rate. It seems that the use of such codes does not decrease the security of the McEliece cryptosystem [JM96].

7.5 Negative Results for Cryptographic Primitives

In this section we show that cryptographic tasks which require some form of "non-malleability" cannot be performed by functions with low input locality. This includes MACs, signatures and non-malleable encryption schemes (e.g., CCA2 secure encryptions). We prove our results in the private-key setting (i.e., for MAC and symmetric encryption). This makes them stronger as any construction that gains security in the public-key setting is also secure in the private-key setting. We will also prove that there is no PRG in Local_2 and therefore the results of Section 7.4.2 are optimal.

7.5.1 Basic Observations

Let $f : \{0,1\}^n \to \{0,1\}^{s(n)}$ be a function and let $s = s(n)$. For $i \in [n]$ and $x \in \{0,1\}^n$, we let $Q_i(x) \subseteq [s]$ be the set of indices in which $f(x)$ and $f(x_{\oplus i})$ differ. (Recall that $x_{\oplus i}$ denote the string x with the i-th bit flipped.) We let $Q_i^n \overset{\text{def}}{=} \bigcup_{x\in\{0,1\}^n} Q_i(x)$, equivalently, Q_i^n is the set of output bits which are affected by the i-th input bit. From now on, we omit the superscript n whenever the input length is clear from the context. We show that, given an oracle access to f, we can efficiently approximate the set Q_i for every i.

Lemma 7.5.1 *There exists a probabilistic algorithm A that, given an oracle access to $f : \{0,1\}^n \to \{0,1\}^s$, an index $i \in [n]$ and an accuracy parameter ε, outputs a set $Q \subseteq Q_i$ such that $\Pr_x[Q_i(x) \nsubseteq Q] \leq \varepsilon$, where the probability is taken over the coin tosses of A and the choice of x (which is independent of A). Moreover, when $f : \{0,1\}^n \to \{0,1\}^{s(n)}$ is an infinite function the time complexity of A is polynomial in $n, s(n)$ and $\varepsilon^{-1}(n)$. In particular, if $s(n) = \text{poly}(n)$, then for every constant c, one can reduce the error to n^{-c} in time $\text{poly}(n)$.*

Proof: Let $t = \ln(2s/\varepsilon)$ and $\alpha = \varepsilon/(2s)$. The algorithm A constructs the set Q iteratively, starting from an empty set. In each iteration, A chooses uniformly and independently at random a string $x \in \{0,1\}^n$ and adds to Q the indices for which $f(x)$ and $f(x_{\oplus i})$ differ. After t/α iterations A halts and outputs Q. Clearly, $Q \subseteq Q_i$. Let $p_j \overset{\text{def}}{=} \Pr_x[j \in Q_i(x)]$. We say that j is common if $p_j > \alpha$. Then, if j is common we have

$$\Pr[j \notin Q] \leq (1-p_j)^{t/\alpha} \leq (1-\alpha)^{t/\alpha} \leq \exp(-t) = \varepsilon/(2s).$$

Since $|Q_i| \leq s$, there are at most s common j's and thus, by a union bound, the probability that A misses a common j is at most $\varepsilon/2$. On the other hand, for a random x, the probability that $Q_i(x)$ contains an uncommon index is at most $s \cdot \alpha = \varepsilon/2$. Hence, we have $\Pr_x[Q_i(x) \nsubseteq Q] \leq \varepsilon$, which completes the proof. ■

Our negative results are based on the following simple observation.

Lemma 7.5.2 *Let $f : \{0,1\}^n \to \{0,1\}^{s(n)}$ be a function in $\text{Local}_{l(n)}$. Then, there exist a probabilistic polynomial-time algorithm A such that for every $x \in \{0,1\}^n$ and $i \in [n]$, the output of A on $(y = f(x), i, Q_i^n, 1^n)$ equals, with probability at least $2^{-l(n)}$, to the string $y' = f(x_{\oplus i})$. In particular, when $l(n) = O(\log(n))$, the success probability of A is $1/\text{poly}(n)$.*

Proof: Fix n and let $s = s(n)$ and $Q_i = Q_i^n$. By definition, y and y' may differ only in the indices of Q_i. Hence, we may randomly choose y' from a set of size $2^{|Q_i|} \leq 2^{l(n)}$, and the lemma follows. ■

Note that the above lemma generalizes to the case in which, instead of getting the set Q_i^n, the algorithm A gets a set Q'_i that satisfies $Q_i(x) \subseteq Q'_i \subseteq Q_i^n$.

By combining the above lemmas we get the following corollary:

Corollary 7.5.3 *Let $f : \{0,1\}^n \to \{0,1\}^{s(n)}$ be a function in $\mathrm{Local}_{l(n)}$, where $s(n) = \mathrm{poly}(n)$. Then, there exist a probabilistic polynomial-time algorithm A that, given an oracle access to f, converts, with probability $(1-1/n)\cdot 2^{-l(n)}$, an image $y = f(x)$ of a randomly chosen string $x \leftarrow U_n$ into an image $y' = f(x_{\oplus 1})$. Namely,*

$$\Pr_x[A^f(f(x), 1^n) = f(x_{\oplus 1})] \geq (1 - 1/n) \cdot 2^{-l(n)},$$

where the probability is taken over the choice of x and the coin tosses of A. In particular, when $l(n) = O(\log(n))$ the algorithm A succeeds with probability $1/\mathrm{poly}(n)$,

Proof: First, we use algorithm A_1 of Lemma 7.5.1 to learn, with accuracy $\varepsilon = 1/n$, an approximation Q'_1 of the set Q_1^n. Then, we invoke the algorithm A_2 of Lemma 7.5.2 on $(f(x), 1, Q'_1, 1^n)$ where $f(x)$ is the challenge given to us. By the above lemmas, we get,

$$\Pr_x[A^f(f(x), 1^n) = f(x_{\oplus 1})] \geq \Pr_x[Q_1(x) \subseteq Q'_1] \cdot \Pr_x[A_2(f(x), 1, Q'_1, 1^n) = f(x_{\oplus 1})] \geq (1 - 1/n) \cdot 2^{-l(n)},$$

and the corollary follows. ■

Clearly, one can choose to flip any input bit and not just the first one. Also, we can increase the success probability to $(1 - n^{-c}) \cdot 2^{-l(n)}$ for any constant c.

We now prove the impossibility results.

7.5.2 MACs and Signatures

Let (S, V) be a MAC scheme, where the randomized signing function $S(k, \alpha, r)$ computes a signature β on the document α using the (random and secret) key k and randomness r, and the verification algorithm $V(k, \alpha, \beta)$ verifies that β is a valid signature on α using the key k. The scheme is secure (unforgeable) if it is infeasible to forge a signature in a chosen message attack. Namely, any efficient adversary that gets an oracle access to the signing process $S(k, \cdot)$ fails to produce a valid signature β on a document α (with respect to the corresponding key k) for which it has not requested a signature from the oracle.[5] The scheme is one-time secure if the adversary is allowed to query the signing oracle only once.

Suppose that the signature function $S(k, \alpha, r)$ has logarithmic input locality (i.e., $S(k, \alpha, r) \in \mathrm{Local}_{O(\log(|k|))}$). Then, we can use Corollary 7.5.3 to break the scheme with a single oracle call. First, ask the signing oracle $S(k, \cdot)$ to sign on a randomly chosen document α. Then, use the algorithm of Corollary 7.5.3 to transform, with probability $1/\mathrm{poly}(n)$, the valid pair (α, β) we received from the signing oracle into a valid pair $(\alpha_{\oplus 1}, \beta')$. (Note that when applying Corollary 7.5.3 we let $S(\cdot, \cdot, \cdot)$ play the role of f.)

[5]When querying the signing oracle, the adversary chooses only the message and is not allowed to choose the randomness which the oracle uses to produce the signature.

Now, suppose that for each *fixed* key k the signature function $S_k(\alpha, r) = S(k, \alpha, r)$ has input locality $\ell(n)$. In this case we cannot use Corollary 7.5.3 directly. The problem is that we cannot apply Lemma 7.5.1 to learn the set Q_i (i.e., the set of output bits which are affected by the i-th input bit of $f = S_k(\cdot, \cdot)$) since we do not have a full oracle access to S_k. (In particular, we do not see or control the randomness used in each invocation of S_k.) However, we can guess the set Q_i and then apply Lemma 7.5.2. This attack succeeds with probability $(1/\binom{s(n)}{\ell(n)}) \cdot 2^{-\ell(n)}$ where $s(n)$ is the length of the signature (and so is polynomial in n). When $\ell(n) = c$ is constant, the success probability is $1/\Theta(s(n)^c) = 1/\text{poly}(n)$ and therefore, in this case, we break the scheme.[6] To summarize:

Theorem 7.5.4 *Let (S, V) be a MAC scheme. If $S(k, \alpha, r) \in \text{Local}_{O(\log(|k|))}$ or $S_k(\alpha, r) \in \text{Local}_{O(1)}$ for every k, then the scheme is not one-time secure.*

Remarks on Theorem 7.5.4.

1. Theorem 7.5.4 is true even if *some* bit of α has low input locality. This observation also holds in the case of non-malleable encryption scheme.
2. If we have an access to the verification oracle (for example, in the public-key setting), we can break the scheme in a stronger sense. Specifically, we can forge a signature to *any* target document given a single signature to, say, 0^n. To see this note that, given a signature β of the document α, we can *deterministically* find a signature β' of the document $\alpha_{\oplus i}$ by checking all the polynomially many candidates. Hence, we can apply this procedure $O(n)$ times and gradually transform a given signature of some arbitrary document into a signature of any target document.
3. A *weaker* version of Theorem 7.5.4 still holds even when the input locality of the signing algorithm is *logarithmic* with respect to any *fixed* key (i.e., when $S_k(\alpha, r) \in \text{Local}_{O(\log(|k|))}$ for every k). In particular, we can break such MAC schemes assuming that we are allowed to ask for several signatures that were produced with some fixed (possibly unknown) randomness. In such a case, we use Lemma 7.5.1 to (approximately) learn the output bits affected by, say, the first input bit, and then apply Lemma 7.5.2 to break the scheme. This attack rules out the existence of a *deterministic* MAC scheme for which $S_k(\alpha) \in \text{Local}_{O(\log(|k|))}$ for every k.

7.5.3 Non-Malleable Encryption

Let (E, D) be a private-key encryption scheme, where the encryption function $E(k, m, r)$ computes a ciphertext c encrypting the message m using the (random and secret) key k and randomness r, and the decryption algorithm $D(k, c, r)$ decrypts the ciphertext c that was encrypted under the key k. Roughly speaking, non-malleability of an encryption scheme guarantees that it is infeasible to modify a ciphertext c into a ciphertext c' of a message related to the decryption of c.

Theorem 7.5.5 *Let (E, D) be a private-key encryption scheme. If $E(k, m, r) \in \text{Local}_{O(\log(|k|))}$ or $E_k(m, r) \in \text{Local}_{O(1)}$ for every k, then the scheme is malleable with respect to an adversary that has no access to neither the encryption oracle nor the decryption oracle. If (G, E, D) is a public-key encryption scheme and $E_k(m, r) \in \text{Local}_{O(\log(|k|))}$ for every k, then the scheme is malleable.*

[6]When the locality $\ell(n)$ of S_k is logarithmic (for every fixed key k), this approach yields an attack that succeeds with probability $1/n^{\Theta(\log(n))}$.

Proof: The proof is similar to the proof of case of Theorem 7.5.4. Let n be the length of the key k, $p = p(n), m = m(n)$, and $s = s(n)$ be the lengths of the message x, randomness r, and ciphertext length c respectively; i.e., $E : \{0,1\}^n \times \{0,1\}^p \times \{0,1\}^m \to \{0,1\}^s$. Our attacks will use the message space $M = \{0,1\}^p$ and the relation R for which $(x, x') \in R$ if and only if x and x' differ only in their first bit.

Suppose that the encryption function $E(k, x, r)$ has logarithmic input locality (i.e., $E(k, x, r) \in \mathrm{Local}_{O(\log(|k|))}$). Then, by Corollary 7.5.3, we can break the scheme by transforming, with noticeable probability, the challenge ciphertext c into a ciphertext c' such that the corresponding plaintexts differ only in their first bit. Clearly, the probability for this relation to hold with respect to $\tilde{c}$ which is a ciphertext of a random plaintext is negligible. Hence, we break the scheme.

Now, suppose that for each fixed key k the encryption function $E_k(x, r) = E(k, x, r)$ has input locality $\ell(n)$. In this case we guess the set Q_1 and then apply Lemma 7.5.2. This attack succeeds with probability $(1/\binom{s(n)}{\ell(n)}) \cdot 2^{-\ell(n)}$. When $\ell(n)$ is constant, the success probability is $1/\mathrm{poly}(n)$ and therefore, in this case, the scheme is broken.

We move on to the case in which the input locality of E_k is logarithmic. The previous attack succeeds in this case with probability $1/n^{\Theta(\log(n))}$. However, we can improve this to $1/\mathrm{poly}(n)$ if we have a *stronger* access to the encryption oracle. In particular, we should be able to get several ciphertexts that were produced with some fixed (possibly unknown) randomness. In such a case, we use Lemma 7.5.1 to (approximately) learn the output bits affected by, say, the first input bit, and then apply Lemma 7.5.2 to break the scheme. The public-key setting is a special case in which this attack is feasible as we get a full access to the randomness of the encryption oracle. ■

7.5.4 The Impossibility of Implementing a PRG in Local_2

We prove that there is no PRG in Local_2 and thus the PRG constructed in Section 7.4.2 has optimal input locality as well as optimal output locality.

Lemma 7.5.6 *Let $G : \{0,1\}^n \to \{0,1\}^{s(n)}$ be a polynomial-time computable function in Local_2 where $s(n) > n$. Then, there exists a polynomial-size circuit family $\{A_n\}$ that given $z \in \mathrm{Im}(G)$ reads some subset $S \subset [s(n)]$ of z's bits and outputs z_k for some $k \notin S$.*

Proof: Fix n and let $s = s(n)$. Define $H_G = ((\mathrm{Out} = [s], \mathrm{In} = [n]), E)$ to be the bipartite graph whose edges correspond to the input-output dependencies in G; that is, (i, j) is an edge if and only if the i-th output bit of G depends on the j-th input bit. Since G is in Local_2 the average output degree is at most $2n/s$ which is smaller than 2 (as $s > n$). The circuit A_n implements the following procedure:

1. Initialize S to be an empty set.

2. If there exists an output $k \in \mathrm{Out}$ which is not connected to any input, then halt and predict z_k according to z_S (and G).

3. Otherwise, there exists an output $j \in \mathrm{Out}$ which depends on a single input bit $i \in \mathrm{In}$ (since the average out degree is smaller than 2). Add i to S and remove i and j from the graph.

4. goto 2

The procedure stops after at most n steps since there are only $n < s$ inputs. The correctness follows from the fact that the output bit k depends only on the input bits indexed by S. ■

We can now conclude that there is no PRG in Local_2.

Corollary 7.5.7 *There is no PRG in* Local_2.

Proof: Assume, towards a contradiction, that $G : \{0,1\}^n \rightarrow \{0,1\}^{s(n)}$ is a PRG in Local_2. Fix n and let $s = s(n)$. Let A_n be the adversary defined in Lemma 7.5.6. Then, we define an adversary B_n that given $z \in \{0,1\}^s$ invokes A_n and checks whether A_n predicts z_k correctly. By Lemma 7.5.6, when $z \in \mathrm{Im}(G)$ the adversary B_n always outputs 1. However, when z is a random string the probability that B_n outputs 1 is at most 1/2. Indeed, if $B_n(z) = 1$ for some $z \in \{0,1\}^s$ then $B_n(z_{\oplus k}) = 0$, where k is the bit that A_n predicts when reading z (and $z_{\oplus i}$ denote the string z with the i-th bit flipped). Hence, A_n errs on at least half of the strings in $\{0,1\}^s$, and so it distinguishes $G(U_n)$ from U_s with advantage 1/2. ■

The above corollary can be extended to rule out the existence of a *collection* of PRGs with input locality 2. Note that when G is chosen from a collection, the graph H_G might not be available to the adversary constructed in Lemma 7.5.6. However, a closer look at this lemma, shows that, in fact, the adversary does not need an explicit description of H_G; rather, it suffices to find an approximation of H_G (in the sense of Lemma 7.5.1). As shown in Lemma 7.5.1, such an approximation can be found efficiently (with, say, $1/n$ error probability) given an (oracle) access to G. This modification also shows that such a PRG can be broken by a *uniform* adversary.

7.6 Negative Results for Randomized Encodings

In the following, we prove some negative results regarding randomized encoding with low input locality. In Section 7.6.1, we provide a necessary condition for a function to have such an encoding. We use this condition to prove that some simple (NC^0) functions cannot be encoded by functions having sub-linear input locality (regardless of the complexity of the encoding). This is contrasted with the case of constant output locality, where it is known that *every* function f can be encoded by a function $\hat{f}$ whose output locality is 4 and whose complexity is polynomial in the size of the branching program that computes f (see Section 4.2). In Section 7.6.2 we show that, although linear functions do admit efficient constant-input encoding, they do not admit an efficient *universal* constant-input encoding. That is, one should use different decoders and simulators for each linear function. This is contrasted with previous constructions of randomized encoding with constant output locality (cf. Section 4.2) which gives a (non-efficient) universal encoding for the class of all functions $f : \{0,1\}^n \rightarrow \{0,1\}^l$ as well as an efficient universal encoding for classes such as all linear functions or all size-s BPs (where s is polynomial in n).

7.6.1 A Necessary Condition for Encoding with Low Input Locality

Let $f : \{0,1\}^n \rightarrow \{0,1\}^l$ be a function. Define an undirected graph G_i over $\mathrm{Im}(f)$ such that there is an edge between the strings y and y' if there exists $x \in \{0,1\}^n$ such that $f(x) = y$ and $f(x_{\oplus i}) = y'$. Let $\hat{f} : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^s$ be a (perfectly correct and private) randomized encoding of f with decoder B and simulator S. Let $Q_i \subseteq \{1, \ldots, s\}$ be the set of output bits in $\hat{f}$ which are affected by the input variable x_i. Namely, $j \in Q_i$ iff $\exists x \in \{0,1\}^n, r \in \{0,1\}^m$ such that the strings $\hat{f}(x,r)$ and $\hat{f}(x_{\oplus i}, r)$ differ on the j-th bit.

We need the following claims.

Claim 7.6.1 *Let $y, y' \in \mathrm{Im}(f)$ be adjacent vertices in G_i. Then, for every $\hat{y} \in \mathrm{support}(S(y))$ there exists $\hat{y}' \in \mathrm{support}(S(y'))$ which differs from $\hat{y}$ only in indices which are in Q_i.*

Proof: Let $x \in \{0,1\}^n$ be an input string for which $f(x) = y$ and $f(x_{\oplus i}) = y'$. Fix some $\hat{y} \in \mathrm{support}(S(y))$. Then, by perfect privacy, there exists some $r \in \{0,1\}^m$ for which $\hat{y} = \hat{f}(x,r)$. Let $\hat{y}' = \hat{f}(x_{\oplus i}, r)$. By the definition of Q_i, the strings $\hat{y}$ and $\hat{y}'$ differ only in indices which are in Q_i. Also, since $\hat{f}$ is perfectly private $\hat{y}' \in \mathrm{support}(S(y'))$, and the claim follows. ■

Claim 7.6.2 *Let $y \in \mathrm{Im}(f)$ and let $\hat{y} \in \mathrm{Im}(\hat{f})$. Then, $y = B(\hat{y})$ if and only if $\hat{y} \in \mathrm{support}(S(y))$.*

Proof: Let $x \in f^{-1}(y)$. By perfect correctness, $y = B(\hat{y})$ iff $\hat{y} \in \mathrm{support}(\hat{f}(x, U_m))$. By perfect privacy, $\mathrm{support}(\hat{f}(x,U_m)) = \mathrm{support}(S(f(x))) = \mathrm{support}(S(y))$, and the claim follows. ■

We can now prove the following lemma.

Lemma 7.6.3 *The size of each connected component of G_i is at most $2^{|Q_i|}$.*

Proof: Let $Q = Q_i$ and $G = G_i$. Fix $u \in \mathrm{Im}(f)$ and let $\hat{u} \in \{0,1\}^s$ be some arbitrary element of $\mathrm{support}(S(u))$. Let $Z \stackrel{\mathrm{def}}{=} \{z \in \{0,1\}^s \mid z_i = \hat{u}_i, \forall i \in [s] \setminus Q\}$. That is, $z \in Z$ if it differs from $\hat{u}$ only in indices which are in Q. To prove the claim, we define an onto mapping from Z (whose cardinality is $2^{|Q|}$) to the members of the connected component of u. The mapping is defined by applying the decoder B of $\hat{f}$, namely $z \to B(z)$. (Assume, wlog, that if the decoder is invoked on a string z which is not in $\mathrm{Im}(\hat{f})$ then it outputs $\perp$.) Let $v \in \mathrm{Im}(f)$ be a member of the connected component of u. We prove that there exists $z \in Z$ such that $v = B(z)$.

The proof is by induction on the distance (in edges) of v from u in the graph G. In the base case when the distance is 0, we let $z = \hat{u}$ and, by perfect correctness, get that $B(\hat{u}) = u$. For the induction step, suppose that the distance is $t+1$. Then, let w be the last vertex in a shortest path from u to v. By the induction hypothesis, there exists a string $\hat{w} \in Z$ for which $w = B(\hat{w})$. Hence, by Claim 7.6.2, $\hat{w} \in \mathrm{support}(S(w))$. Since u and w are neighbors, we can apply Claim 7.6.1 and conclude that there exists $\hat{v} \in \mathrm{support}(S(v))$ which differs from $\hat{w}$ only in indices which are in Q. Since $\hat{w} \in Z$ it follows that $\hat{v}$ is also in Z. Finally, by Claim 7.6.2, we have that $B(\hat{v}) = v$ which completes the proof. ■

Corollary 7.6.4 *A function $f : \{0,1\}^n \to \{0,1\}^l$ can be perfectly encoded by a function $\hat{f} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$ in Local_t only if for every $1 \le i \le n$ the size of the connected components of G_i is at most 2^t.*

The above corollary shows that even some very simple functions do not admit an encoding with constant input locality. Consider, for example, the function

$$f(x_1, \ldots, x_n) = x_1 \cdot (x_2, \ldots, x_n) = (x_1 \cdot x_2, x_1 \cdot x_3, \ldots, x_1 \cdot x_n).$$

For every $y \in \mathrm{Im}(f) = \{0,1\}^{n-1}$ it holds that $f(1,y) = y$ and $f(0,y) = 0^{n-1}$. Hence, every vertex in G_1 is a neighbor of 0^{n-1} and the size of the connected component of G_1 is 2^{n-1}. Thus, the input locality of x_1 in any perfect encoding of this function is $n-1$. (Note that this matches the results of Section 7.3 since $\mathrm{r}ank(x_1) = n-1$.)

7.6.2 Impossibility of Universal Encoding for Linear Functions

For a class C of functions that map n-bits into l-bits, we say that C has a universal encoding in the class $\hat{C}$ if there exists a universal simulator S and a universal decoder B such that, for every function $f_z \in C$, there is an encoding $\hat{f}_z \in \hat{C}$ which is private and correct with respect to the simulator S and the decoder B.

We show that, although linear functions do admit encodings with constant input locality, they do not admit such a *universal* encoding. Recall that the set of affine functions L (mapping n bits to l bits) is a family of pairwise independent hash functions and thus can be used (unconditionally) as a one-time secure MAC. Suppose that this class of functions H had a universal encoding with constant input locality. Then, by the results of Section 4.8.1, we would have a one-time secure MACs (S, V) whose signing algorithm has constant input locality for every fixed key; i.e., $S_k(\alpha, r) \in \text{Local}_{O(1)}$ for every fixed key k. However, the results of Section 7.5.2 rule out the existence of such a scheme. We now give a more direct proof to the impossibility of obtaining a universal encoding with constant input locality for linear functions. The proof is similar to the proofs in Section 7.6.1.

Let C be a class of functions that map n bits into l bits. For each input bit $1 \leq i \leq n$, we define a graph G_i over $\cup_{f \in C}\text{Im}(f)$ such that there is an edge between the strings y and y' if there exists $x \in \{0,1\}^n$ and $f \in C$ such that $f(x) = y$ and $f(x_{\oplus i}) = y'$. Namely, $G_i = \cup_{f \in C} G_i(f)$, where $G_i(f)$ is the graph defined in Section 7.6.1. Suppose that C has universal encoding in Local_t with decoder B and simulator S. That is, for every $f_z \in C$ there exists a perfect randomized encoding $\hat{f} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$ in Local_t whose correctness and privacy hold with respect to B and S.

Lemma 7.6.5 *The degree of every vertex in G_i is bounded by $\binom{s}{t} \cdot 2^t$.*

Proof: Let y be a vertex of G_i and fix some $\hat{y} \in S(y)$. Let y' be a neighbor of y with respect to $f \in C$. Let $Q = Q_i \subseteq \{1, \ldots, s\}$ be the set of output bits in $\hat{f}$ which are affected by the input variable x_i. Then, by Lemma 7.6.3, there exists a set $Z_Q \subseteq \{0,1\}^s$ of size 2^t such that $y' \in \text{Im}(B(Z_Q))$. Hence, we have an onto mapping from $Q \times Z_Q$ to the neighbors of y. Thus, the number of neighbors is at most $\binom{s}{t} \cdot 2^t$. ■

Lemma 7.6.6 *Let C be the class of linear functions $L : \{0,1\}^n \to \{0,1\}^l$ where $l \leq n$. Then, for every $1 \leq i \leq n$ the graph G_i is a complete graph over $\{0,1\}^l$.*

Proof: Consider, for example, the graph $G = G_1$ and fix some $y, y' \in \{0,1\}^l$. Then, for $\sigma = (0, 1, \ldots, 1)$ and $\sigma_{\oplus 1} = (1, 1, \ldots, 1)$, there exists a linear function $L : \{0,1\}^n \to \{0,1\}^l$ for which $y = L(\sigma)$ and $y' = L(\sigma_{\oplus 1})$. To see this, write L as a matrix $M \in \{0,1\}^{l \times n}$ such that $L(x) = Mx$. Let $M = (M_1, M')$, that is M_1 denotes the leftmost column of M, and M' denotes the matrix M without M_1. Now, we can first solve the linear system $M \cdot \sigma = y$ which is equivalent to $M' \cdot \sigma = y$ and then solve the linear system $M \cdot \sigma_{\oplus 1} = y'$ which is now equivalent to $M_1 = y' - y$. ■

Let $l \leq n$. By combining the above claims we conclude that the output complexity s of any universal encoding in Local_t for linear functions $L : \{0,1\}^n \to \{0,1\}^l$ must satisfy $\binom{s}{t} \cdot 2^t \geq 2^l$. In particular, when t is constant, the output complexity of the encoding must be exponential in l.

7.7 Conclusions and Open Questions

We showed that, under standard intractability assumptions, cryptographic primitives such as OWFs, PRGs and public-key encryption can be computed by functions of constant input locality. On the other hand, primitives that require some form of “non-malleability”, such as MACs, signatures and non-malleable encryption schemes, cannot be computed by such functions. An interesting open question is whether collision-resistent hash functions can be realized by functions of constant input locality. It is not hard to see that such functions are extremely vulnerable to near-collision attacks (since if x, x' are “close” in Hamming distance, then so are their images $h(x)$ and $h(x')$). However, it is not clear whether this weakness allows to find actual collisions.

Chapter 8

One-Way Functions with Optimal Output Locality

Summary: In Chapter 4 it was shown that, under relatively mild assumptions, there exist one-way functions (OWFs) in NC^0_4. This result is not far from optimal as as there is no OWF in NC^0_2. The gap is partially closed in Chapter 7 by showing that the existence of a OWF (and even a PRG) in NC^0_3 is implied by the intractability of decoding a random linear code. In this chapter we provide further evidence for the existence of OWF in NC^0_3. We construct such a OWF based on the existence of a OWF that enjoys a certain strong "robustness" property. Specifically, we require that the adversary cannot invert f even if it is given, in addition to the output $y = f(x)$, all bits of x influencing a randomly chosen subset of the bits of y. We also show how to construct such a function assuming that a random function of locality $O(\log n)$ is one-way. (A similar assumption was previously made by Goldreich [Gol00].) The transformation from "robust" OWF to OWF in NC^0_3 is obtained by constructing a new variant of randomized encoding.

8.1 Introduction

The results presented so far leave a small gap between the strong positive evidence for cryptography in NC^0_4 and the known impossibility of even OWF in NC^0_2.[1] In this chapter we attempt to close this gap for the case of OWF, providing positive evidence for the existence of OWF in NC^0_3.

A natural approach for closing the gap would be to reduce the degree of our general construction of randomized encodings from 3 to 2. (Indeed, Construction 4.2.4 transforms a degree-2 encoding into one in NC^0_3.) However, the results of [IK00] provide some evidence against the prospects of this general approach, ruling out the existence of degree-2 perfectly private encodings for most nontrivial functions (see Section 3.3.3). Thus, we may take the following two alternative approaches: (1) seek *direct* constructions of degree-2 OWF based on specific intractability assumptions; and (2) employ a relaxed variant of randomized encodings which enables a degree-2 representation of general functions.

In Chapter 7, we used approach (1) to construct a OWF (and even a PRG) with optimal locality based on the presumed intractability of decoding a random linear code. In this chapter we

[1] There is a similar gap in terms of algebraic degree: we have positive results for degree-3 cryptography, whereas degree-1 cryptography is clearly impossible.

demonstrate the usefulness of approach (2) by employing degree-2 randomized encodings with a weak (but nontrivial) privacy property that we call *semi-privacy.* This encoding allows to construct a OWF with optimal locality based on a OWF that enjoys a certain strong "robustness" property. We also show how to construct such a robust OWF assuming that a random function of logarithmic locality is one-way (a similar assumption was suggested in [Gol00]). We stress that our approach does not yield a general result in the spirit of the results of Chapter 4. Thus, we happen to pay for optimal degree and locality with the loss of generality.

8.1.1 Semi-Private Randomized Encoding and Robust OWF

Let $\hat{f}$ be a randomized encoding of f. Recall that, according to the privacy property, the output distribution of $\hat{f}(x,r)$ (induced by a uniform choice of r) should hide all the information about x except for the value $f(x)$. Semi-privacy relaxes this requirement by insisting that the input x remain hidden by $\hat{f}(x,r)$ only in the case that $f(x)$ takes some specific value, say 0. (If $f(x)$ is different from this value, $\hat{f}(x,r)$ fully reveal x.) As it turns out, this relaxed privacy requirement is sufficiently liberal to allow a degree-2 encoding of general boolean functions.

Given any OWF f, one could attempt to apply a semi-private encoding as described above to every output bit of f, obtaining a degree-2 function $\hat{f}$. However, $\hat{f}$ will typically not be one-way: every output bit of f that evaluates to 1 might reveal the entire input. This motivates the following notion of a *robust* OWF. Loosely speaking, a OWF f is said to be robust if it remains (slightly) hard to invert even if a random subset of its output bits are "exposed", in the sense that all input bits leading to these outputs are revealed. More specifically, consider the following inversion game. First, we choose a random input $x \in \{0,1\}^n$, compute $f(x)$ and send it to the adversary. Then, for each output bit of f we toss a coin y_i. If $y_i = 1$, we allow the adversary to see the bits of x that influence the i-th output bit. That is, we send $(x_{K(i)}, i, y_i)$ to the adversary, where $K(i) \subseteq [n]$ is the set of inputs that affects the i-th output bit. If $y_i = 0$, we reveal nothing regarding x and send (i, y_i) to the adversary. The adversary wins the game if he finds a preimage x' which is consistent with the game, i.e., $f(x') = f(x)$, and $x'_{K(i)} = x_{K(i)}$ whenever $y_i = 1$. The function is robust one-way if, for some polynomial $p(\cdot)$, any efficient adversary fails to find a consistent preimage with probability $1/p(n)$. (A formal definition is given in Section 8.2.)

Intuitively, the purpose of the robustness requirement is to guarantee that the information leaked by the semi-private encoding leaves enough uncertainty about the input to make inversion difficult. Indeed, we show that when semi-private randomized encoding (SPRE) is applied to a (slightly modified) robust OWF the resulting function is weakly one-way. Hence, a construction of degree-2 SPRE can be used to convert a robust OWF to OWF with degree 2. However, we fail to achieve such a construction. Instead, we get a weak variant of SPRE. Fortunately, it turns out that this variant still gives a degree-2 distributionally one-way function, which can be transformed into standard OWF with degree 2.

8.1.2 Constructing Robust OWF

We construct a robust OWF under the assumption that a random function of logarithmic locality is one-way. More specifically, we rely on the following intractability assumption. Chose a random function $f : \{0,1\}^n \to \{0,1\}^n$ of logarithmic locality by choosing a random predicate P_i for each output bit, and a random input-output bipartite graph $G = ((\text{Out} = [n], \text{In} = [n]), E)$ whose average output degree is $d = O(\log n)$. (That is, $f_i(x) = P_i(x_{E(i)})$ where $E(i)$ is the set of inputs that touch the i-th output of G). Then, we assume that, for some polynomial $p(\cdot)$, any efficient adversary that gets the description of f fails to invert it (on a randomly chosen input) with probability at least $1/p(n)$. Namely, we assume that the above procedure defines a collection of weakly OWFs. (See [Gol01a, Definition 2.4.3].) A similar assumption was proposed by Goldreich in [Gol00], where it is shown that if the graph G satisfies some expansion property, then the function resists a natural class of inverting attacks. Since a random graph has good expansion with high probability, our variant resists the same class of attacks.

We show that, under the above assumption, a randomly chosen function $h : \{0,1\}^{n\cdot\exp(d)} \to 2n$ with (average) locality d/n is robust one-way. Intuitively, h is an extension of f to a bigger graph such that, with non-negligible probability, the random exposure of h, is similar to f; thus, it is slightly hard to invert and h is robust one-way.

8.2 Preliminaries

In this section we give some definitions and prove a useful lemma.

8.2.1 Semi-Private Randomized Encoding

The following definition relaxes the notion of randomized encoding.

Definition 8.2.1 (Semi-private randomized encoding (SPRE)) *Let $f : \{0,1\}^n \to \{0,1\}$ be a boolean function. We say that a function $\hat{f} : \{0,1\}^n \times \{0,1\}^{m(n)} \to \{0,1\}^{s(n)}$ is a* semi-private randomized encoding *(SPRE) of f if the following conditions hold:*

- **Perfect-correctness.** *There exists a (possibly inefficient) decoder B, such that for every $x \in \{0,1\}^n$ it holds that $B(1^n, \hat{f}(x, U_{m(n)})) = f(x)$.*
- **One-sided privacy.** *There exists a probabilistic polynomial time simulator S_0, such that for every $x \in f^{-1}(0) \cap \{0,1\}^n$, it holds that $S_0(1^n) \equiv \hat{f}(x, U_{m(n)})$.*
- **One-sided exposure.** *There exists a (possibly inefficient) exposure algorithm E_1, such that for every $x \in f^{-1}(1) \cap \{0,1\}^n$ and every $r \in \{0,1\}^{m(n)}$, it holds that $E_1(1^n, \hat{f}(x,r)) = x$.*

In Section 8.3 we will show how to construct a (relaxed version) of SPRE of degree 2 for non-trivial class of functions.

8.2.2 One-Way Functions

We review several variants of one-way functions (OWFs).

Definition 8.2.2 (One-way function) *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a function. Then,*

- **Hard to invert.** *The function f is* hard to invert *if for every (non-uniform) polynomial-time algorithm, A, the probability* $\Pr[A(1^n, f(U_n)) \in f^{-1}(f(U_n))]$ *is negligible in n.*
- **Slightly hard to invert.** *The function f is* slightly hard to invert *if there exists a polynomial $p(\cdot)$, such that for every (non-uniform) polynomial-time algorithm, A, and all sufficiently large n's* $\Pr[A(1^n, f(U_n)) \notin f^{-1}(f(U_n))] > \frac{1}{p(n)}$.

- **Distributionally hard to invert.** *The function f is* distributionally hard to invert *if there exists a positive polynomial $p(\cdot)$ such that for every (non-uniform) polynomial-time algorithm, A, and all sufficiently large n's,* $\mathrm{SD}((A(1^n, f(U_n)), f(U_n)), (U_n, f(U_n))) > \frac{1}{p(n)}$.

If f can be computed in polynomial-time and it is also hard to invert (resp. slightly hard to invert, distributionally hard to invert) then it is called one-way *(resp.* weakly one-way*,* distributionally one-way*).*

Note that the first variant defined above is the standard notion of OWF. The following lemma shows how to transform a degree 2 distributionally one-way function into a (standard) OWF with degree 2.

Lemma 8.2.3 *A degree-2 distributional OWF implies a degree-2 OWF in* NC^0_3.

Proof: First observe that a degree-2 weak OWF can be transformed into a degree-2 (standard) OWF (cf. [Yao82],[Gol01a, Theorem 2.3.2]). Combined with Construction 4.2.4), we get that the existence of a degree-2 weak OWF implies the existence of a degree-2 OWF in NC^0_3. Hence it is enough to show how to transform a degree-2 distributional OWF into a degree-2 weak OWF.

Let f be a degree-2 distributional OWF. Consider the function $F(x, i, h) = (f(x), h_i(x), i, h)$, where $x \in \{0,1\}^n$, $i \in \{1, \ldots, n\}$, $h : \{0,1\}^n \to \{0,1\}^n$ is a pairwise independent hash function, and h_i denotes the i-bit-long prefix of $h(x)$. This function was defined by Impagliazzo and Luby [IL89], who showed that in this case F is weakly one-way (see also [Gol01a, p. 96]). Note that $h(x)$ can be computed as a degree-2 function of x and (the representation of) h by using the hash family $h_{M,v}(x) = xM + v$, where M is an $n \times n$ matrix and v is a vector of length n. However, $h_i(x)$ is not of degree 2 when considered as a function of h, x and i, since "chopping" the last $n - i$ bits of $h(x)$ raises the degree of the function when i is not fixed. We get around this problem by applying n copies of F on independent inputs, where each copy uses a different i. Namely, we define the function $F'((x^{(i)}, h^{(i)})_{i=1}^n) \stackrel{\mathrm{def}}{=} (F(x^{(i)}, i, h^{(i)}))_{i=1}^n$. Since each of the i's is now fixed, the resulting function F' can be computed by degree-2 polynomials over $\mathbb{F}_2$. Moreover, it is not hard to verify that F' is weakly one-way if F is weakly one-way. We briefly sketch the argument. Given an efficient inverting algorithm A for F', one can invert $y = F(x, i, h) = (f(x), h_i(x), i, h)$ as follows. For every $j \neq i$, uniformly and independently choose $x^{(j)}, h^{(j)}$, set $z_j = F(x^{(j)}, j, h^{(j)})$ and $z_i = y$, then invoke A on $(z_j)_{j=1}^n$ and output the i-th block of the answer. This inversion algorithm for F has the same success probability as A on a polynomially related input. ■

We proceed with the definition of robust one-way function.

Definition 8.2.4 (Robust one-way function) *Let $f : \{0,1\}^n \to \{0,1\}^{l(n)}$ be a polynomial-time computable function. Let f_i be the boolean function computing the i-*th *bit of f. Let $K(i) \stackrel{def}{=} \{1 \leq j \leq n | f_i$ depends on $x_j\}$. Define the function $f_{\exp}(x, b) \stackrel{def}{=} (f_i(x), b_i, b_i \wedge x_{K(i)})_{i=1}^{l(n)}$, where $x \in \{0,1\}^n, b \in \{0,1\}^{l(n)}$, the string $x_{K(i)}$ is the restriction of x to the indices in $K(i)$, and $b_i \wedge x_{K(i)}$ denotes an AND between the bit b_i and every bit of the string $x_{K(i)}$. We refer to the function $f_{\exp}$ as the* random exposure *of $f(x)$. Then, f is called a* robust OWF *if its random exposure $f_{\exp}$ is a weak OWF.*

8.3 From Robust OWF to OWF in NC^0_3

8.3.1 A Warmup

We now argue that a degree-2 implementation of SPRE would suffice to transform a robust OWF f into a degree-2 (weak) OWF h. This transformation is done in two steps. First, we generate a function g by masking the output bits of f with distinct random bits y_i; the bits y_i are also included in the output of g. Thus, each of the output bits of g is 1 with probability $\frac{1}{2}$. Moreover, inverting the random exposure of f is as difficult as inverting g given an exposure of all of its outputs that evaluate to 1. Finally, we construct a function h that outputs an SPRE of each output bit of g along with the masks y_i.[2] Intuitively, each output bit of g is exposed by the SPRE only if it evaluates to 1, and therefore an adversary inverting h can be used to invert the random exposure of f. We will now formalize this intuition.

Lemma 8.3.1 (Encoding Robust OWF via SPRE) *Let $f : \{0,1\}^n \to \{0,1\}^{l(n)}$ be a robust one-way function. Let $\{\hat{g}_i((x, y_i), r_i)\}_{i=1}^{l(n)}$ be a uniform SPRE of the collection $\{g_i(x, y_i) \overset{def}{=} f_i(x) \oplus y_i\}_{i=1}^{l(n)}$ where f_i computes the i-th output bit of f and $y_i \in \{0,1\}$. Define the function*

$$h(x, y, (r_1, \ldots, r_{l(n)})) \overset{def}{=} (\hat{g}_i((x_{K(i)}, y_i), r_i), y_i)_{i=1}^{l(n)},$$

where $x \in \{0,1\}^n, y \in \{0,1\}^{l(n)}$, and r_i is the randomness needed by $\hat{g}_i$. Then, the function h is weakly one-way.

Proof: Denote by $f_{\exp}$ the random exposure of f. Since the input lengths of $f, f_{\exp}, \hat{g}$ are are all polynomials in n we can express our claims in terms of n. Let $p(n)$ be the polynomial guaranteed by the assumption that $f_{\exp}$ is weakly one-way. Namely, every (non-uniform) polynomial-time algorithm fails to invert $f_{\exp}$ on the uniform distribution with probability at least $\frac{1}{p(n)}$. Assume, towards a contradiction, that h is not weakly one-way. It follows that there exists an efficient algorithm $\hat{A}$ that inverts h on infinitely many n's with probability greater than $1 - \frac{1}{p(n)}$. We construct an efficient algorithm A that inverts $f_{\exp}$ with similar success, in contradiction to the robustness of f.

For $1 \leq i \leq l(n)$ let $a_i, b_i \in \{0,1\}$ and $c_i \in \{0,1\}^{n_i}$, where n_i denotes the number of bits that affect f_i. Then, on the input $(a_i, b_i, c_i)_{i=1}^{l(n)}$ (supposedly in the range of $f_{\exp}(x, b)$; that is, $a_i = f_i(x)$ and $c_i = b_i \wedge x_{K(i)}$, for some x) the algorithm A does the following:

1. For each $1 \leq i \leq l(n)$,
   (a) if $b_i = 0$ (that is the i-th output bit of f was not exposed) set $y_i = a_i$, invoke $S_0(1^n, 1^i)$ the perfect one-sided simulator of $\hat{g}_i$, and record the result in z_i.
   (b) if $b_i = 1$ (that is, c_i is supposed to be the exposure of the i-th output bit of f) set $y_i = a_i \oplus 1$, compute $\hat{g}_i((c_i, y_i), r_i)$, where r_i is chosen from the uniform distribution, and record the result in z_i.
2. Invoke $\hat{A}$ on $(z_i, y_i)_{i=1}^{l(n)}$ and output (x', b), where $(x', y', r'_1, \ldots, r'_{l(n)})$ is the output of $\hat{A}$.

[2]Were the mask bits not given in the output, an adversary could have easily inverted h by computing f on an input x and selecting corresponding masks.

We now argue that A inverts f_{exp} whenever $\hat{A}$ inverts h.

Claim 8.3.2 *Suppose that $(a_i, b_i, c_i)_{i=1}^{l(n)}$ is in the range of f_{exp}, and $(x', y', r'_1, \dots, r'_{l(n)}) \in h^{-1}((z_i, y_i)_{i=1}^{l(n)})$, where $(z_i, y_i)_{i=1}^{l(n)}$ are generated by A as described above. Then $(x', b) \in {f_{\mathrm{exp}}}^{-1}((a_i, b_i, c_i)_{i=1}^{l(n)})$.*

Proof: Since $(a_i, b_i, c_i)_{i=1}^{l(n)}$ is in the range of f_{exp}, it suffices to prove that for each $1 \le i \le l(n)$ if $b_i = 0$ then $f_i(x'_{K(i)}) = a_i$ and if $b_i = 1$ then $x'_{K(i)} = c_i$. Let $(x', y', r'_1, \dots, r'_{l(n)}) \in h^{-1}((z_i, y_i)_{i=1}^{l(n)})$; thus, for every i, it holds that $\hat{g}_i((x'_{K(i)}, y'_i), r'_i) = z_i$ and $y'_i = y_i$.

If $b_i = 0$ then, by the definition of A, it holds that $\hat{g}_i((x'_{K(i)}, a_i), r'_i) \in S_0(1^n, 1^i)$ (since $y'_i = y_i = a_i$ and $z_i \in S_0(1^n, 1^i)$). By the perfect privacy and the perfect correctness, $g_i(x'_{K(i)}, a_i) = 0$. As $g_i(\alpha, \beta) = f_i(\alpha) \oplus \beta$, we get $f_i(x'_{K(i)}) = a_i$.

If $b_i = 1$ then $\hat{g}_i((x'_{K(i)}, a_i \oplus 1), r'_i) = \hat{g}_i((c_i, a_i \oplus 1), r_i)$ since $y'_i = y_i = a_i \oplus 1$. By the perfect correctness and the definition of g_i, it holds that, $g_i(x'_{K(i)}, a_i \oplus 1) = g_i(c_i, a_i \oplus 1) = f_i(c_i) \oplus a_i \oplus 1 = 1$, where the last equality follows from $f_i(c_i) = a_i$. However, since $\hat{g}_i$ exposes $x_{K(i)}$ for every x such that $g_i(x) = 1$, it follows that $x'_{K(i)} = c_i$, which completes the proof. $\square$

We now claim that $\hat{A}$ is invoked on the appropriate probability distribution. Formally,

Claim 8.3.3 *Suppose that A is invoked on $f_{\mathrm{exp}}(x, b)$, where x, b are uniformly distributed. Then, $(z_i, y_i)_{i=1}^{l(n)} \equiv h(x', y', r_1, \dots, r_{l(n)})$ where $x', y', r_1, \dots, r_{l(n)}$ are uniformly distributed.*

Proof: Let $i \in \{1, \dots, l(n)\}$. If $b_i = 0$ then $g_i(x_{K(i)}, y_i) = f_i(x_{K(i)}) \oplus y_i = f_i(x_{K(i)}) \oplus f_i(x_{K(i)}) = 0$ since $y_i = f_i(x_{K(i)})$. Thus, by the perfect privacy, it holds that $z_i \equiv S(1^n, 1^i) \equiv \hat{g}_i((x_{K(i)}, y_i), r_i)$, where r_i is uniformly distributed. If $b_i = 1$ then $z_i \equiv \hat{g}_i((x_{K(i)}, y_i), r_i)$, where r_i is uniformly distributed. Together, we get that, $(z_i, y_i)_{i=1}^{l(n)} \equiv h(x, y, r_1, \dots, r_{l(n)})$ where $x, r_1, \dots, r_{l(n)}$ are uniformly distributed. Note that $y_i = f_i(x_{K(i)}) \oplus b_i$; thus, since b_i is uniformly distributed, so is y_i, and the claim follows. $\square$

Combining the previous two claims together we get that A inverts f_{exp} with probability greater than $1 - \frac{1}{p(n)}$ in contradiction to our hypothesis. $\blacksquare$

8.3.2 The Actual Construction

We turn to the question of constructing degree-2 SPRE for general functions. We do not know how to construct a degree-2 SPRE in the strict sense of Definition 8.2.1. However, we can construct a *statistical* variant of such an encoding that suffices for our purposes. Since we will only need to encode functions that depend on $d = O(\log n)$ inputs, it will be convenient to construct such an encoding based on the DNF representation of the function.

Construction 8.3.4 (Relaxed SPRE for canonic DNF) *Let $g : \{0,1\}^d \to \{0,1\}$ be a boolean function. Let $\bigvee_{i=1}^{k} T_i$ be its unique canonic DNF representation. That is, for each α such that $g(\alpha) = 1$ there exists a corresponding term T_i which evaluates to 1 iff $x = \alpha$. We encode such T_i by the degree-2 function $\hat{T}_i(x, r) = \langle x - \alpha, r \rangle$, where $\langle \cdot, \cdot \rangle$ denotes inner product over $\mathbb{F}_2$. Let t*

be some integer (later used as a security parameter). Then, the degree-2 function $\hat{g}$ is defined by concatenating t copies of $\hat{T}_i$ (each copy with independent random variables $r_{i,j}$) for each of the k terms. Namely,

$$\hat{g}(x, (r_{i,j})_{1 \leq i \leq k, 1 \leq j \leq t}) \stackrel{def}{=} ((\hat{T}_1(x, r_{1,j}))_{j=1}^t, \ldots, (\hat{T}_k(x, r_{k,j}))_{j=1}^t),$$

where $r_{i,j} \in \{0,1\}^d$.

Claim 8.3.5 *Let $g : \{0,1\}^d \to \{0,1\}$ be a boolean function whose canonic DNF contains k terms. Construct $\hat{g}$ from g as in the previous construction. Then, there exists an SPRE $\hat{g}'$ of g such that* $\Pr[\hat{g}'(x, r) \neq \hat{g}(x, r)] \leq k \cdot 2^{-t}$*, where t denotes the security parameter of $\hat{g}$ and the probability is taken over $x \leftarrow U_d, r \leftarrow U_{d \cdot t \cdot k}$. Moreover, the running time of the one-sided simulator of $\hat{g}'$ is* $\mathrm{poly}(tk)$.

Proof: Let $\bigvee_{i=1}^k T_i$ be the canonic DNF representation of g. We view the variables $r_{i,j}$ of $\hat{T}$ as the random input of the encoding. Observe that if $T_i(x) = 1$ then $\hat{T}_i(x, r) = 0$ for every r. On the other hand, if $T_i(x) = 0$ then $\hat{T}_i(x, U_d)$ is distributed uniformly over $\mathbb{F}_2$ (since $\hat{T}_i$ is an inner product of a random vector with a non-zero vector). Therefore, if $g(x) = 0$, then the output of all the copies of each $\hat{T}_i$ are distributed uniformly and independently over $\mathbb{F}_2$ (since we used independent random variables). If $g(x) = 1$ then there exist a single term T_i that equals to one and the other terms equal to zero (since this is a canonic DNF); thus all the copies of $\hat{T}_i$ equal to zero while the other $\hat{T}_j$'s are distributed uniformly and independently over $\mathbb{F}_2$.

We define the function $\hat{g}'$ by slightly modifying $\hat{g}$ in the following way: $\hat{g}'(x, r)$ equals to $\hat{g}(x, r)$ if for every term T_i that is not satisfied by x, there exist at least one copy of $\hat{T}_i$ that equals to 1; Otherwise, for each i such that $T_i(x) = 0$ and $(\hat{T}_i(x, r_{i,j}))_{j=1}^t = 0^t$, we replace all the copies of $\hat{T}_i$ by 1. Since each copy of $\hat{T}_i$ is distributed uniformly when $T_i(x) = 0$, it holds that $\Pr_{r \leftarrow U_{d \cdot t \cdot k}}[\hat{g}'(x, r) \neq \hat{g}(x, r)] \leq k \cdot 2^{-t}$ for every $x \in \{0,1\}^d$; hence, $\Pr_{x \leftarrow U_d, r \leftarrow U_{d \cdot t \cdot k}}[\hat{g}'(x, r) \neq \hat{g}(x, r)] \leq k \cdot 2^{-t}$.

We now define a simulator, decoder and exposing algorithms, and show that $\hat{g}'$ is a semi-private randomized encoding of g. The simulator S_0 independently chooses k strings $y_1, \ldots, y_k$ each from the distribution U_t; for each y_i that equals to the all-zero string, S_0 replaces it with the all-one string and outputs $y_1, \ldots, y_k$. Hence, S_0 runs in time $\mathrm{poly}(tk)$. Clearly, for every $x \in g^{-1}(0) \cap \{0,1\}^d$ the simulator S_0 perfectly simulates the distribution of $\hat{g}'(x, U_{d \cdot t \cdot k})$.

On input $y = y_1, \ldots, y_k$, where $y_i \in \{0,1\}^t$, the decoder B outputs 1 if there exists an i such that $y_i = 0^t$; otherwise, it outputs 0. Let $y = \hat{g}'(x, r)$; by the definition of $\hat{g}'$, we have $y_i = 0^t$ if and only if $T_i(x) = 1$. Thus, the decoder never errs.

Finally, the exposing algorithm E_1, given $y = y_1, \ldots, y_k$, finds the first i such that $y_i = 0^t$ and outputs the unique assignment x' that satisfies the term T_i; otherwise, E_1 outputs some arbitrary value, say the zero string. For $x \in g^{-1}(1) \cap \{0,1\}^d$, there exists exactly one (canonic) term such that $T_i(x) = 1$ (again, since we use a canonic DNF). By the definition of $\hat{g}'$, only the string y_i that corresponds to this term is the all-zero string, and thus the exposing algorithm never errs (when x satisfies g). ■

Lemma 8.3.6 (Robust OWF to distributional OWF via relaxed SPRE) *Let $f : \{0,1\}^n \to \{0,1\}^{l(n)}$ be a robust OWF. Let f_i be the function that computes the i-th bit of f. Suppose that* *the locality of f is $O(\log n)$, furthermore assume that the truth table of each f_i can be computed in polynomial time. Define the degree-2 function*

$$h(x, y, (r_1, \ldots, r_{l(n)})) \stackrel{def}{=} (\hat{g}_i((x_{K(i)}, y_i), r_i), y_i)_{i=1}^{l(n)},$$

where $\hat{g}$ is the DNF-based encoding of $g_i(x, y_i) \stackrel{def}{=} f_i(x) \oplus y_i$ defined in Construction 8.3.4 and the security parameter satisfies $\omega(\log n) < t(n) < \text{poly}(n)$. Then, the function h is distributionally one-way.

Proof: Let $\hat{g}'_i$ be the (non-relaxed) SPRE which is statistically close to $\hat{g}_i$ (promised by Claim 8.3.5). Define h' similarly to h where $\hat{g}_i$ is replaced with $\hat{g}'_i$. Then, by Lemma 8.3.1, h' is slightly hard to invert (as in the definition of weak OWF). Note that the number of terms in the canonic DNF of each g_i is bounded by $\text{poly}(n)$ since the locality of f is $O(\log n)$. Hence we can write,

$$\begin{aligned}\Pr[h(x, y, r) \neq h'(x, y, r)] &\leq \sum_{i=1}^{l(n)} \Pr[\hat{g}_i((x_{K(i)}, y_i), r_i) \neq \hat{g}'_i((x_{K(i)}, y_i), r_i)] \\ &\leq l(n) \cdot \text{poly}(n) \cdot 2^{-t(n)} \leq \varepsilon(n),\end{aligned}$$

where $\varepsilon(n)$ is negligible in n and $x, y, r = r_1, \ldots, r_{l(n)}$ are uniformly distributed. The first inequality follows from the union bound, the second inequality follows from Claim 8.3.5 and third inequality follows from $\omega(\log n) < t(n)$.

Observe that h is efficiently computable since the truth tables of the g_i's are computable in polynomial time and since $t(n) < \text{poly}(n)$. Hence, we can apply Lemma 4.4.4 to h, h' and deduce that h is distributionally one-way. ■

Combining Lemma 8.2.3 and Lemma 8.3.6 we derive the following theorem.

Theorem 8.3.7 *Let $f : \{0,1\}^n \to \{0,1\}^{l(n)}$ be a robust OWF whose locality is $O(\log n)$ and the truth table of each of its output bits can be computed in polynomial time. Then, there exist a degree-2 OWF in NC^0_3.*

8.4 A Candidate Robust One-Way Function

Denote by $G_p^{l,m}$ the distribution over bipartite undirected graphs $((U, V), E)$ having l vertices on one side and m vertices on the other side, $U = \{1, \ldots, l\}, V = \{1, \ldots, m\}$, in which each of the lm potential edges between U and V exists with probability p. The vertices of U, V are also referred to as input vertices and output vertices respectively.

Let $d(n) = O(\log n)$ be an average degree parameter. For our assumption we use a graph $G = ((U, V), E)$ that is selected from $G_{d/n}^{n,n}$; namely a bipartite graph with n vertices in each side where each of its potential n^2 edges exists with probability d/n (and hence the expected degree of every vertex is d). We use $E(i) \subseteq U$ to denote the set of vertices in U that are connected to the vertex $i \in V$. Let $Q = (Q_1, \ldots, Q_n)$ where $Q_i : \{0,1\}^{2d} \to \{0,1\}$ is a random predicate whose truth table is defined by 2^{2d} random bits, and let $x \in \{0,1\}^n$. Define the function

$$f_{G,Q}(x) \stackrel{\text{def}}{=} (Q_1(x_{E(1)}), \ldots, Q_n(x_{E(n)})).$$

That is, given an n-bit string x the function $f_{G,Q}$ outputs, for each $i \in \{1, \ldots, n\}$, the value $Q_i(x_{E(i)})$. Since the size of $E(i)$ does not necessarily match the domain size of Q_i, we use a prefix of $x_{E(i)}$ if $|E(i)| > 2d$, or pad it with a suffix of zeros if $|E(i)| < 2d$. We assume that the collection of functions $f_{G,Q}$ is weakly one-way.[3] Loosely speaking, this means that when the graph G (that defines the input-output dependencies) and the predicates Q_i (that define the exact relation between the i-th output bit and the corresponding input bits) are picked from the appropriate probability distribution, then inverting the resulting function is hard. Formally,

Intractability Assumption 8.4.1 (Random graph) *There exist (an efficiently computable) function $d(n) = O(\log n)$ and a polynomial $p(\cdot)$, such that for every (non-uniform) polynomial-time algorithm, A, and all sufficiently large n's*

$$\Pr[A(1^n, G, Q, f_{G,Q}(x)) \notin f_G^{-1}(f_{G,Q}(x))] > \frac{1}{p(n)},$$

where x and Q are uniformly distributed and G is selected from $G_{d/n}^{n,n}$.

The function $f_{G,Q}$ used in Assumption 8.4.1 is a variant of a candidate OWF proposed by Goldreich [Gol00] (the main difference being that [Gol00] uses the same predicate for all output bits). It is shown in [Gol00] that if the graph G satisfies some expansion property, then the function resists a natural class of inverting attacks. Since a random graph has good expansion with high probability, our variant resists the same class of attacks.

Based on Assumption 8.4.1, we present a collection of robust OWFs h that meets the requirements of Theorem 8.3.7. (A collection of functions is robust one-way, if its random exposure is a weakly one-way collection.) Intuitively, h is an extension of $f_{G,Q}$ to a bigger graph such that, with non-negligible probability, the function $h_{\exp}$, which is the random exposure of h, is similar to $f_{G,Q}$; thus, $h_{\exp}$ is slightly hard to invert and h is robust one-way.

Lemma 8.4.2 *Let $d = O(\log n)$ be the function guaranteed by Assumption 8.4.1. Define the collection of functions*

$$h_{H,P}(x) \overset{def}{=} (P_1(x_{E(1)}), \ldots, P_{2n}(x_{E(2n)})),$$

where x is an $n \cdot \exp(d)$-bit string, P is a vector of $2n$ predicates $P_i : \{0,1\}^{2d} \to \{0,1\}$, and $H = ((U,V), E)$ is a bipartite graph selected from $G_{d/n}^{n\cdot\exp(d),2n}$. (Here $\exp(\cdot)$ denotes the natural exponential function.) If Assumption 8.4.1 holds, then $h_{H,P}$ is a robust one-way collection.

Proof: Let $n_f(n), n_h(n)$ be the input lengths of $f_{G,Q}, h_{H,P}$ respectively. (Note that $d = O(\log n)$ and therefore $n_f(n), n_h(n)$ are polynomials in n.) Let $p(n)$ be the polynomial guaranteed by the assumption that $f_{G,Q}$ is weakly one-way. Namely, every non-uniform polynomial-time algorithm fails to invert $f_{G,Q}$ on $U_{n_f(n)}$ with probability at least $\frac{1}{p(n)}$. Assume, towards a contradiction, that the random exposure of $h_{H,P}$ is not weakly one-way. It follows that, for every polynomial $q(n)$, there exists an efficient algorithm $\hat{A}$ that for infinitely many n's inverts the random exposure of $h_{H,P}$ with success probability $1 - \frac{1}{q(n)}$. We show that for $q(n) > p(n)n^{1.5} \cdot \exp(d) = \text{poly}(n)$, we

[3] For our purpose, a collection of OWFs is a collection of functions parameterized with a public key that is selected according to a predefined (efficiently-constructible) distribution and is given to the adversary along with the challenge. More specifically, we refer here to a *public-coins* collection. (See Appendix A). For a general definition, see [Gol01a, Definition 2.4.3].

can use algorithm $\hat{A}$ to construct an efficient algorithm A that inverts $f_{G,Q}$ on infinitely many n's with success probability greater than $1 - \frac{1}{p(n)}$, in contradiction to Assumption 8.4.1. Algorithm A gets as an input the string $(G = ((U, V), E), Q, y_1, \ldots, y_n)$ (supposedly $y_i = Q_i(x_{E(i)})$ for some $x \in \{0,1\}^n$) and proceeds as follows:

1. Let $S = \{n+1, \ldots, n \cdot \exp(d)\}$ and $T = \{n+1, \ldots, 2n\}$. Select uniformly and independently each of the possible edges between S and $V \cup T$ with probability d/n. Denote the set of selected edges by D. Let H denote the bipartite graph $((U \cup S, V \cup T), E \cup D)$.
2. Uniformly choose an n-bit string z, and n predicates $P_{n+1}, \ldots, P_{2n}$ where $P_i : \{0,1\}^{2d} \rightarrow \{0,1\}$. For each $i \in V$, if $|E(i)| < 2d$ then for every $|E(i)|$-bit string a, swap the value of Q_i on $(a \circ z_{D(i)})$ with the value of Q_i on $(a \circ 0^{2d-|E(i)|})$, where $\circ$ is the concatenation operator. Denote this new predicate by $P_i : \{0,1\}^{2d} \rightarrow \{0,1\}$ (if $|E(i)| \geq 2d$ then P_i is equal to Q_i).
3. Invoke the algorithm $\hat{A}$ on $((y_i, 0, 0^{|E(i)|})_{i=1}^{n}, (P_j(z_{D(j)}), 1, z_{D(j)})_{j=n+1}^{2n}$ with (H, P) as the key of h. Output x', where x' is the n-bit-long prefix of $\hat{A}$'s answer (x', z').

That is, algorithm A augments the given graph G into a bigger graph H, whose exposed outputs are the vertices in T, and their exposed inputs are the vertices of S, namely the value of z.

Claim 8.4.3 *If $\hat{A}$ succeeds in inverting its input, then A also succeeds in inverting its input.*

Proof: Suppose that $\hat{A}$ succeeds in inverting its input, then $y_i = P_i(x'_{E(i)}, z'_{D(i)})$ and $z'_{D(i)} = z_{D(i)}$. If $2d \leq |E(i)|$ then $y_i = P_i(x'_{E(i)}, z'_{D(i)}) = P_i(x'_{E(i)}) = Q_i(x'_{E(i)})$, since $U = \{1, \ldots, n\}$. On the other hand, if $|E(i)| < 2d$ then, by the definition of P_i, it holds that $y_i = P_i(x'_{E(i)}, z'_{D(i)}) = P_i(x'_{E(i)}, z_{D(i)}) = Q_i(x'_{E(i)}, 0^{2d-|E(i)|}) = Q_i(x'_{E(i)})$. Therefore, x' itself is a pre-image of y_i under Q_i and G. □

Let $(G = ((U, V), E), Q, (Q_i(x_{E(i)}))_{i=1}^{n})$ be the input given to A, where $x \leftarrow U_n$, $G \leftarrow G_{d/n}^{n \cdot \exp(d), 2n}$, and $Q \leftarrow U_{2n2^{2d}}$. Recall that $\hat{A}$ is guaranteed to succeed with high probability when it is invoked on a uniformly distributed input; i.e., when the predicates and the inputs are chosen uniformly, each of the potential $2n \cdot n \cdot \exp(d)$ edges of H exist with probability d/n and each of the outputs of the graph is exposed with probability $\frac{1}{2}$. Observe that the predicates P_i are indeed distributed uniformly (since they were constructed by permuting the rows of the randomly chosen truth tables of the Q_i's), and so are the input strings x, z. Nevertheless, it seems that the graph we give $\hat{A}$ and its exposure are not distributed as they should be. In particular, the unexposed vertices of $G \cup H$ are exactly the first n output vertices. To fix this, we add another step to algorithm A, in which A applies two randomly chosen permutations, one on the input vertices of H and one on its output vertices. We denote this new graph by H'. The input permutation should be also applied to the truth tables of the predicates and to the strings x, z; similarly, A permutes the predicates order according to the output permutation (the resulting predicates are still distributed uniformly). This way, Claim 8.4.3 still holds.

Denote by F the "correct" input distribution of $\hat{A}$; that is, F is a graph selected from $G_{d/n}^{n \cdot \exp(d), 2n}$ where each of its output vertices is exposed with probability $\frac{1}{2}$. Observe that each of the potential edges of H' exists independently with probability d/n, except the edges between the exposed outputs T', and the unexposed inputs U', that do not exist. Then, the distribution of the exposed graph $H' = ((U' \cup S', V' \cup T'), E' \cup D')$ is equivalent to that of F conditioned on the event that there are exactly n unexposed outputs in F and exactly n inputs that are not connected to the exposed outputs; namely, that there are n unexposed outputs and n unexposed inputs. Denote this event by GOOD_n. Clearly,

$$\begin{aligned}\Pr[\mathrm{GOOD}_n] =& \Pr[n \text{ outputs were not exposed}] \\ & \cdot \Pr[n \text{ inputs were not exposed} | n \text{ outputs were not exposed}].\end{aligned}$$

Since each of the $2n$ output vertices is exposed with probability $\frac{1}{2}$, we get

$$\Pr[n \text{ outputs were not exposed}] = \binom{2n}{n} \cdot 2^{-2n} = \Omega(\frac{1}{\sqrt{n}}).$$

Recall that an input vertex is not exposed if it is not connected to an exposed output vertex. If exactly n output vertices were not exposed, then exactly n output vertices were exposed. Given this event, the probability that an input vertex is not exposed is exactly $(1 - d/n)^n$ since each edge exists with probability d/n. Then, $\Pr[n \text{ inputs were not exposed}|n \text{ outputs were exposed}] = b(n; n \cdot \exp(d), (1 - d/n)^n)$, where $b(k; m, p)$ is the binomial distribution; that is, the probability to have exactly k successes out of a sequence of m independent Bernoulli trails, each with a probability p of success. Note that $b(\lfloor mp \rfloor; m, p) > \frac{1}{m+1}$, since $\sum_{k=0}^{m} b(k; m; p) = 1$ and since the binomial distribution viewed as a function of k is maximized when k equals to the expectation, that is when $k = \lfloor pm \rfloor$. Therefore, for sufficiently large n's it holds that

$$\Pr[n \text{ inputs were not exposed}|n \text{ outputs were exposed}] \geq \frac{1}{n \cdot \exp(d)},$$

since $\lim(1 - d/n)^n = \exp(-d)$ as $n \to \infty$. Combining together we get that, $\Pr[\mathrm{GOOD}_n] > 1/(n^{1.5} \cdot \exp(d))$. Also we have, $\Pr[A \text{ succeeds}] \geq \Pr[\hat{A} \text{ succeeds}|\mathrm{GOOD}_n]$, where A and $\hat{A}$ are invoked on $f_{G,Q}(U_{n_{f}(n)})$, and on a random exposure of $h_{H,P}(U_{n_{h}(n)})$ respectively. Thus we have,

$$\begin{aligned}1 - \frac{1}{q(n)} < \Pr[\hat{A} \text{ succeeds}] &= \Pr[\hat{A} \text{ succeeds}|\mathrm{GOOD}_n] \Pr[\mathrm{GOOD}_n] \\ &\quad + \Pr[\hat{A} \text{ succeeds}|\overline{\mathrm{GOOD}}_n](1 - \Pr[\mathrm{GOOD}_n]) \\ &\leq \Pr[\hat{A} \text{ succeeds}|\mathrm{GOOD}_n] \Pr[\mathrm{GOOD}_n] + 1 - \Pr[\mathrm{GOOD}_n],\end{aligned}$$

which implies,

$$\begin{aligned}\Pr[\hat{A} \text{ succeeds}|\mathrm{GOOD}_n] &> \left(\Pr[\mathrm{GOOD}_n] - \frac{1}{q(n)}\right) \frac{1}{\Pr[\mathrm{GOOD}_n]} \\ &= 1 - \frac{1}{q(n) \Pr[\mathrm{GOOD}_n]} > 1 - \frac{n^{1.5} \cdot \exp(d)}{q(n)}.\end{aligned}$$

Since $q(n) > p(n)n^{1.5} \cdot \exp(d)$, we get that A succeeds with probability greater than $1 - \frac{1}{p(n)}$, and the lemma follows. ■

Since $d = O(\log n)$, the locality of $h_{H,P}$ is $O(\log n)$ also the truth tables P_i are given explicitly in the description of $h_{H,P}$. Thus, we can apply Theorem 8.3.7 and construct a collection of degree-2 OWFs in NC_3^0 based on Assumption 8.4.1.

Theorem 8.4.4 *Under Assumption 8.4.1, there exists a collection of degree-2 OWF in* NC_3^0.

Appendix A

On Collections of Cryptographic Primitives

In most cases, we view a cryptographic primitive (e.g., a OWF or a PRG) as a single function $f : \{0,1\}^* \to \{0,1\}^*$. However, it is often useful to consider more general variants of such primitives, defined by a *collection* of functions $\{f_z\}_{z \in Z}$, where $Z \subseteq \{0,1\}^*$ and each f_z is defined over a finite domain D_z. The full specification of such a collection usually consists of a probabilistic polynomial time key-generation algorithm that chooses an index z of a function (given a security parameter 1^n), a domain sampler algorithm that samples a random element from D_z given z, and a function evaluation algorithm that computes $f_z(x)$ given z and $x \in D_z$. The primitive should be secure with respect to the distribution defined by the key-generation and the domain sampler. (See a formal definition for the case of OWF in [Gol01a, Definition 2.4.3].)

Collections of primitives arise naturally in the context of parallel cryptography, as they allow to shift "non-parallelizable" operations such as prime number selection and modular exponentiations to the key-generation stage (cf. [NR04]). They also fit naturally into the setting of P-uniform circuits, since the key-generation algorithm can be embedded in the algorithm generating the circuit. Thus, it will be convenient to assume that z is a description of a circuit computing f_z. When referring to a collection of functions from a given complexity class (e.g., NC^1,NC^0_4, or $\mathcal{PREN}$, cf. Definition 3.1.9) we assume that the key generation algorithm outputs a description of a circuit from this class. In fact, one can view collections in our context as a natural relaxation of uniformity, allowing the circuit generator to be randomized. (The above discussion also applies to other P-uniform representation models we use, such as branching programs.)

Our usage of collections differs from the standard one in that we insist on D_z being the set of *all* strings of a given length (i.e., the set of all possible inputs for the circuit z) and restrict the domain sampler to be a trivial one which outputs a uniformly random string of the appropriate length. This convention guarantees that the primitive can indeed be invoked with the specified parallel complexity, and does not implicitly rely on a (possibly less parallel) domain sampler.[1] In most cases, it is possible to modify standard collections of primitives to conform to the above convention. We illustrate this by outlining a construction of an NC^1 collection of one-way permutations based on the intractability of discrete logarithm. The key-generator, on input 1^n, samples a random prime p such that $2^{n-1} \leq p < 2^n$ along with a generator g of Z_p^*, and lets z be a description of an NC^1

[1]Note that unlike the key-generation algorithm, which can be applied "once and for all", the domain sampler should be invoked for each application of the primitive.

circuit computing the function $f_{p,g}$ defined as follows. On an n-bit input x (viewed as an integer such that $0 \leq x < 2^n$) define $f_{p,g}(x) = g^x \mod p$ if $1 \leq x < p$ and $f_{p,g}(x) = x$ otherwise. It is easy to verify that $f_{p,g}$ indeed defines a permutation on $\{0,1\}^n$. Moreover, it can be computed by an NC^1 circuit by incorporating $p, g, g^2, g^4, \ldots, g^{2^n}$ into the circuit. Finally, assuming the intractability of discrete logarithm, the above collection is *weakly* one way. It can be augmented into a collection of (strongly) one-way permutations by using the standard reduction of strong OWF to weak OWF (i.e., using $f'_{p,g}(x_1, \ldots, x_n) = (f_{p,g}(x_1), \ldots, f_{p,g}(x_n))$).

When defining the cryptographic security of a collection of primitives, it is assumed that the adversary (e.g., inverter or distinguisher) is given the key z, in addition to its input in the single-function variant of the primitive. Here one should make a distinction between "private-coin collections", where this is all of the information available to the adversary, and "public-coin collections" in which the adversary is additionally given the internal coin-tosses of the key-generator. (A similar distinction has been recently made in the specific context of collision-resistant hash-functions [HR04]; also, see the discussion of "enhanced TDP" in [Gol04, App. C.1].) The above example for a OWP collection is of the public-coin type. Any public-coin collection is also a private-coin collection, but the converse may not be true.

Summarizing, we consider cryptographic primitives in three different settings:

1. (Single function setting.) The circuit family $\{C_n\}_{n\in\mathbb{N}}$ that computes the primitive is constructed by a deterministic polynomial time circuit generator that, given an input 1^n, outputs the circuit C_n. This is the default setting for most cryptographic primitives.

2. (Public-coin collection.) The circuit generator is a probabilistic polynomial time algorithm that, on input 1^n, samples a circuit from a collection of circuits. The adversary gets as an input the circuit produced by the generator, along with the randomness used to generate it. The experiments defining the success probability of the adversary incorporate the randomness used by the generator, in addition to the other random variables. As in the single function setting, this generation step can be thought of as being done "once and for all", e.g., in a preprocessing stage. Public-coin collections are typically useful for primitives based on discrete logarithm assumptions, where a large prime group should be set up along with its generator and precomputed exponents of the generator.

3. (Private-coin collection.) Same as (2) except that the adversary does not know the randomness that was used by the circuit generator. This relaxation is typically useful for factoring-based constructions, where the adversary should not learn the trapdoor information associated with the public modulus (see [Kha93, NR04]).

We note that our general transformations apply to all of the above settings. In particular, given an NC^1 primitive in any of these settings, we obtain a corresponding NC^0 primitive in the same setting.

Bibliography

[AAR98] M. Agrawal, E. Allender, , and S. Rudich. Reductions in circuit complexity: An isomorphism theorem and a gap theorem. *J. Comput. Syst. Sci.*, 57(2):127–143, 1998.

[ABI86] Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. of Algorithms*, 7(4):567–583, 1986.

[AD97] M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proc. 29th STOC*, pages 284–293, 1997.

[AIK05] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. On one-way functions with optimal locality. Unpublished manuscript available at http://www.cs.technion.ac.il/∼abenny, 2005.

[AIK06a] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *Computional Complexity*, 15(2):115–162, 2006. Preliminary version in Proc. 20th CCC, 2005.

[AIK06b] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. *SIAM J. Comput.*, 36(4):845–888, 2006. Preliminary version in Proc. 45th FOCS, 2004.

[AIK06c] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. On pseudorandom generators with linear stretch in NC^0. In *Proc. 10th Random.*, 2006. Full version to appear in Computational Complexity.

[AIK07] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography with constant input locality. 2007. To appear in Crypto 2007.

[Ajt96] M. Ajtai. Generating hard instances of lattice problems. In *Proc. 28th STOC*, pages 99–108, 1996. Full version in Electronic Colloquium on Computational Complexity (ECCC).

[Ale03] Michael Alekhnovich. More on average case vs approximation complexity. In *Proc. 44th FOCS*, pages 298–307, 2003.

[ALM$^+$98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. *J. of the ACM*, 45(3):501–555, 1998. Preliminary version in Proc. 33rd FOCS, 1992.

[AR94] Noga Alon and Yuval Roichman. Random cayley graphs and expanders. *Random Struct. Algorithms*, 5(2):271–285, 1994.

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. of the ACM*, 45(1):70–122, 1998. Preliminary version in Proc. 33rd FOCS, 1992.

[Bar86] D. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. In *Proc. 18th STOC*, pages 1–5, 1986.

[BFKL94] Avrim Blum, Merrick Furst, Michael Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In *Advances in Cryptology: Proc. of CRYPTO '93*, volume 773 of *LNCS*, pages 278–291, 1994.

[BG85] M. Blum and S. Goldwasser. An efficient probabilistic public-key encryption scheme which hides all partial information. In *Advances in Cryptology: Proc. of CRYPTO '84*, volume 196 of *LNCS*, pages 289–302, 1985.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM*, 50(4):506–519, 2003. Preliminary version in Proc. 32nd STOC, 2000.

[Blu83] M. Blum. Coin flipping by telephone: a protocol for solving impossible problems. *SIGACT News*, 15(1):23–27, 1983.

[BM84] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13:850–864, 1984. Preliminary version in FOCS 82.

[BM92] Mihir Bellare and Silvio Micali. How to sign given any trapdoor permutation. *J. ACM*, 39(1):214–233, 1992. Preliminary version in Proc. 20th STOC, 1988.

[BMR90] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *Proc. of 22nd STOC*, pages 503–513, 1990.

[BNS89] L. Babai, N. Nisan, and Mario Szegedy. Multiparty protocols and logspace-hard pseudorandom sequences. In *Proc. 21st STOC*, pages 1–11, 1989.

[BOGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. of 20th STOC*, pages 1–10, 1988.

[BSSVW03] Eli Ben-Sasson, Madhu Sudan, Salil Vadhan, and Avi Wigderson. Randomness-efficient low-degree tests and short pcps via epsilon-biased sets. In *Proc. 35th STOC*, pages 612–621, 2003.

[BSW03] Boaz Barak, Ronen Shaltiel, and Avi Wigderson. Computational analogues of entropy. In *Proc. 7th Conference on Randomization and Computation, (RANDOM)*, 2003.

[BY96] Mihir Bellare and Moti Yung. Certifying permutations: Noninteractive zero-knowledge based on any trapdoor permutation. *J. of Cryptology*, 9(3):149–166, 1996. Preliminary version in Proc. CRYPTO '92.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd FOCS*, pages 136–145, 2001.

[CCD88] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Proc. of 20th STOC*, pages 11–19, 1988.

[CFIK03] Ronald Cramer, Serge Fehr, Yuval Ishai, and Eyal Kushilevitz. Efficient multi-party computation over rings. In *Proc. EUROCRYPT '03*, pages 596–613, 2003.

[CG88] B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM J. on Computing*, 17(2):230–261, 1988.

[CKN03] R. Canetti, H. Krawczyk, and J. Nielsen. Relaxing chosen ciphertext security of encryption schemes. In *Advances in Cryptology: Proc. of CRYPTO '03*, volume 2729 of *LNCS*, pages 565–582, 2003.

[CM01] M. Cryan and P. B. Miltersen. On pseudorandom generators in NC^0. In *Proc. 26th MFCS*, 2001.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM Press.

[CRVW02] Michael Capalbo, Omer Reingold, Salil Vadhan, and Avi Wigderson. Randomness conductors and constant-degree lossless expanders. In *Proc. 34th STOC*, pages 659–668, 2002.

[Dam88] I.B. Damgård. Collision free hash functions and public key signature schemes. In *Proc. Eurocrypt'87*, pages 203–216, 1988.

[DPP94] I.B. Damgård, T.P. Pedersen, and B. Pfitzmann. On the existence of statistically hiding bit commitment schemes and fail-stop signatures. In *Advances in Cryptology: Proc. of CRYPTO '93*, volume 773 of *LNCS*, pages 250–265, 1994.

[DS05] Yevgeniy Dodis and Adam Smith. Correcting errors without leaking partial information. In *Proc. 37th STOC*, pages 654–663, 2005.

[Fei73] Horst Feistel. Cryptography and computer privacy. *Scientific American*, 228(5), 1973.

[Fei93] J. Feigenbaum. Locally random reductions in interactive complexity theory. In *Advances in Computational Complexity Theory*, volume 13 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages pp. 73–98, 1993.

[Fei02] U. Feige. Relations between average case complexity and approximation complexity. In *Proc. of 34th STOC*, pages 534–543, 2002.

[FLS00] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM J. Comput.*, 29(1):1–28, 2000. Preliminary version in Proc. 31st FOCS, 1990.

[Gam85] T. E. Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in cryptology: Proc. of CRYPTO '84*, volume 196 of *LNCS*, pages 10–18, 1985. or IEEE Transactions on Information Theory, v. IT-31, n. 4, 1985.

[GGH96] O. Goldreich, S. Goldwasser, and S. Halevi. Collision-free hashing from lattice problems. *Electronic Colloquium on Computational Complexity*, 96(042), 1996.

[GGM85] O. Goldreich, S. Goldwasser, and S. Micali. On the cryptographic applications of random functions. In *Advances in Cryptology: Proc. of CRYPTO '84*, volume 196 of *LNCS*, pages 276–288, 1985.

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. of the ACM.*, 33:792–807, 1986.

[GK96] O. Goldreich and A. Kahan. How to construct constant-round zero-knowledge proof systems for NP. *J. of Cryptology*, 9(2):167–189, 1996.

[GKL93] Oded Goldreich, Hugo Krawczyk, and Michael Luby. On the existence of pseudorandom generators. *SIAM J. Comput.*, 22(6):1163–1175, 1993. Preliminary version in Proc. 29th FOCS, 1988.

[GKY89] A. V. Goldberg, M. Kharitonov, and M. Yung. Lower bounds for pseudorandom number generators. In *Proc. 30th FOCS*, pages 242–247, 1989.

[GL89] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *Proc. 21st STOC*, pages 25–32, 1989.

[GM84a] Oded Goldreich and Silvio Micali. Increasing the expansion of pseudorandom generators. Also appears in [Gol01a, Sec. 3.3.2], 1984.

[GM84b] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, 1984. Preliminary version in Proc. STOC '82.

[GMR88] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. on Computing*, 17(2):281–308, 1988.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. Preliminary version in STOC 1985.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game (extended abstract). In *Proc. of 19th STOC*, pages 218–229, 1987.

[Gol90] O. Goldreich. A note on computational indistinguishability. *Inf. Process. Lett.*, 34(6):277–281, 1990.

[Gol98] Oded Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, volume 17 of *Algorithms and Combinatorics*. Springer-Verlag, 1998.

[Gol00] Oded Goldreich. Candidate one-way functions based on expander graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 7(090), 2000.

[Gol01a] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001.

[Gol01b] Oded Goldreich. Randomized methods in computation - lecture notes, 2001. http://www.wisdom.weizmann.ac.il/∼oded/rnd.html.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[GW97] Oded Goldreich and Avi Wigderson. Tiny families of functions with random properties: A quality-size trade-off for hashing. *Random Struct. Algorithms*, 11(4):315–343, 1997.

[Hås87] Johan Håstad. One-way permutations in NC^0. *Information Processing Letters*, 26:153–155, 1987.

[HHR06] Iftach Haitner, Danny Harnik, and Omer Reingold. On the power of the randomized iterate. In *CRYPTO*, pages 22–40, 2006.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[HM96] S. Halevi and S. Micali. Practicle and provably-commitment schemes from collision-free hashing. In *Advances in Cryptology: Proc. of CRYPTO '96*, volume 1109 of *LNCS*, pages 201–215, 1996.

[HR04] Chun Yuan Hsiao and Leonid Reyzin. Finding collisions on a public road, or do secure hash functions need secret coins? In *Advances in Cryptology: Proc. of CRYPTO '04*, volume 3152 of *LNCS*, pages 92–105, 2004.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *Proc. 41st FOCS*, pages 294–304, 2000.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *Proc. 29th ICALP*, pages 244–256, 2002.

[IL89] R. Impagliazzo and M. Luby. One-way functions are essential for complexity based cryptography. In *Proc. of the 30th FOCS*, pages 230–235, 1989.

[Imm88] N. Immerman. Nondeterministic space is closed under complement. *SIAM Journal on Computing*, 17:935–938, 1988. Preliminary version in Proc. 3rd CSCT, 1988.

[IN96] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of Cryptology*, 9:199–216, 1996.

[JM96] Heeralal Janwa and Oscar Moreno. Mceliece public key cryptosystems using algebraic-geometric codes. *Des. Codes Cryptography*, 8(3):293–307, 1996.

[KD79] John B. Kam and George I. Davida. Structured design of substitution-permutation encryption networks. *IEEE Transactions on Computers*, 28(10):747–753, 1979.

[Kha93] Michael Kharitonov. Cryptographic hardness of distribution-specific learning. In *Proc. 25th STOC*, pages 372–381, 1993.

[Kil88] J. Kilian. Founding cryptography on oblivious transfer. In *Proc. 20th STOC*, pages 20–31, 1988.

[KL01] Matthias Krause and Stefan Lucks. On the minimal hardware complexity of pseudorandom function generators (extended abstract). In *Proc. 18th STACS*, volume 2010 of *LNCS*, pages 419–430, 2001.

[KY00] Jonathan Katz and Moti Yung. Complete characterization of security notions for probabilistic private-key encryption. In *STOC*, pages 245–254, 2000.

[Lev73] Leonid A. Levin. Universal sequential search problems. *PINFTRANS: Problems of Information Transmission (translated from Problemy Peredachi Informatsii (Russian))*, 9, 1973.

[LMN93] Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, fourier transform, and learnability. *J. ACM*, 40(3):607–620, 1993. Preliminary version in Proc. 30th FOCS, 1989.

[LP04] Yehuda Lindell and Benny Pinkas. A proof of yao's protocol for secure two-party computation. *Electronic Colloquium on Computational Complexity*, 11(063), 2004.

[Lyu05] Vadim Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In *Proc. 9th Random*, 2005.

[McE78] R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. Technical Report DSN PR 42-44, Jet Prop. Lab., 1978.

[MST03] Elchanan Mossel, Amir Shpilka, and Luca Trevisan. On ϵ-biased generators in NC^0. In *Proc. 44th FOCS*, pages 136–145, 2003.

[Nao91] M. Naor. Bit commitment using pseudorandomness. *J. of Cryptology*, 4:151–158, 1991.

[Nis92] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[NN93] J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856, 1993. Preliminary version in Proc. 22th STOC, 1990.

[NPS99] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proc. 1st ACM Conference on Electronic Commerce*, pages 129–139, 1999.

[NR99] M. Naor and O. Reingold. Synthesizers and their application to the parallel construction of pseudo-random functions. *J. of Computer and Systems Sciences*, 58(2):336–375, 1999.

[NR04] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2):231–262, 2004. Preliminary version in Proc. 38th FOCS, 1997.

[NY89] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proc. of the 21st STOC*, pages 33–43, 1989.

[Ped91] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology: Proc. of CRYPTO '91*, volume 576 of *LNCS*, pages 129–149, 1991.

[PY91] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. of Computer and Systems Sciences*, 43:425–440, 1991. Preliminary version in Proc. 20th STOC, 1988.

[Rab79] M.O. Rabin. Digitalized signatures and public key functions as intractable as factoring. Technical Report 212, LCS, MIT, 1979.

[Reg03] Oded Regev. New lattice based cryptographic constructions. In *Proc. 35th STOC*, pages 407–416, 2003.

[Rog91] Phillip Rogaway. *The Round Complexity of Secure Protocols.* PhD thesis, MIT, June 1991.

[Rom90] J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proc. of the 22nd STOC*, pages 387–394, 1990.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. of the ACM*, 21(2):120–126, 1978.

[RTS00] Jaikumar Radhakrishnan and Amnon Ta-Shma. Tight bounds for depth-two superconcentrators. *SIAM J. Discrete Math.*, 13(1):2–24, 2000. Preliminary version in Proc. 38th FOCS, 1997.

[RTV04] Omer Reingold, Luca Trevisan, and Salil Vadhan. Notions of reducibility between cryptographic primitives. In *TCC '04*, volume 2951 of *LNCS*, pages 1–20, 2004.

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28-4:656–715, 1949.

[Shp06] Amir Shpilka. Constructions of low-degree and error-correcting e-biased generators. In *Proc. 21st Conference on Computational Complexity (CCC)*, pages 33–45, 2006.

[Sud02] Madhu Sudan. Algorithmic introduction to coding theory - lecture notes, 2002. http://theory.csail.mit.edu/∼madhu/FT01/.

[SV03] Amit Sahai and Salil Vadhan. A complete problem for statistical zero knowledge. *J. ACM*, 50(2):196–249, 2003. Preliminary version in FOCS 1997.

[TSUZ01] Amnon Ta-Shma, Christopher Umans, and David Zuckerman. Loss-less condensers, unbalanced expanders, and extractors. In *Proc. 33rd STOC*, pages 143–152, 2001.

[TX03] S. R. Tate and K. Xu. On garbled circuits and constant round secure function evaluation. CoPS Lab Technical Report 2003-02, University of North Texas, 2003.

[Var57] R.R. Varshamov. Estimate of the number of signals in error correcting codes. *Doklady Akademii Nauk SSSR*, 117:739–741, 1957.

[Vio05] Emanuele Viola. On constructing parallel pseudorandom generators from one-way functions. In *Proc. 20th Conference on Computational Complexity (CCC)*, pages 183– 197, 2005.

[Wig94] A. Wigderson. $\mathrm{NL}/poly \subseteq \oplus\mathrm{L}/poly$. In *Proc. 9th Structure in Complexity Theory Conference*, pages 59–62, 1994.

[WT86] A. F. Webster and Stafford E. Tavares. On the design of s-boxes. In *CRYPTO '85: Advances in Cryptology*, pages 523–534, London, UK, 1986. Springer-Verlag.

[Yao82] A. C. Yao. Theory and application of trapdoor functions. In *Proc. 23rd FOCS*, pages 80–91, 1982.

[Yao86] A. C. Yao. How to generate and exchange secrets. In *Proc. 27th FOCS*, pages 162–167, 1986.

[YY94] Xiangdong Yu and Moti Yung. Space lower-bounds for pseudorandom-generators. In *Proc. 9th Structure in Complexity Theory Conference*, pages 186–197, 1994.